Modelo Predictivo para la Asignación Elástica de Recursos sobre Entornos NFV/SDN Basados en OpenStack

Juan Camilo Caviedes Valencia

Universidad Nacional de Colombia
Facultad de Ingeniería
Bogotá D.C, Colombia
2021

Modelo Predictivo para la Asignación Elástica de Recursos sobre Entornos NFV/SDN Basados en OpenStack

Juan Camilo Caviedes Valencia

Tesis o trabajo de grado presentada(o) como requisito parcial para optar al título de:
Magister en Ingeniería - Telecomunicaciones

Director(a):
Ph.D., Luis Fernando Niño Vásquez
Ph.D., Diego Fernando Rueda Pepinosa

Línea de Investigación:
Redes y Sistemas de Telecomunicaciones
Grupo de Investigación:
Laboratorio de Investigación en Sistemas Inteligentes (LISI)

Universidad Nacional de Colombia
Facultad de Ingeniería
Bogotá, D.C, Colombia
2021

A Emma,

Toda nuestra ciencia,
comparada con la realidad,
es primitiva e infantil...
y sin embargo es lo más preciado
que tenemos.

Albert Einstein (1879-1955)

Agradecimientos

Ante todo, quiero agradecer a mi familia por todo el apoyo incondicional que me ha brindado en todos los ámbitos en los que me he desempeñado. Deseo expresar mi más profunda gratitud a mi padre César R. que, a pesar de no llevar su apellido, me brindó todo su amor y apoyo incondicional durante mi formación como persona y, ahora, como magíster. Estoy muy agradecido con mi director PhD. Luis Fernando Niño V., docente de la facultad de ingeniería de la Universidad Nacional y con el grupo de investigación LISI quienes, a través de su insistencia y rigurosidad, inculcaron las bases necesarias para llevar a cabo esta investigación. También, agradecer infinitamente a mi codirector PhD. Diego Fernando Rueda P., líder técnico del área de telecomunicaciones en everis quien, con su vasta experiencia en el sector y su amplia formación académica, me indujo a un tema de investigación apasionante, retador y a la vanguardia de la tecnología en telecomunicaciones. Sin él, no hubiera sido posible abordar en detalle esta investigación. Debo dar un agradecimiento especial a PhD (c) Angie Vega T., por su invaluable ayuda en la redacción y elaboración de este documento. Por último, a Valen, quien siempre estuvo presente durante todo este proceso de formación.

Resumen

La evolución de las tecnologías usadas para mantener la entrega de servicios de calidad sobre redes de telecomunicaciones, ha puesto en evidencia problemáticas alrededor de las arquitecturas con las que han sido concebidas. Principalmente, la diversidad en la forma de conectarse y el despliegue masivo de servicios, ha llevado a la operación de las redes a un punto de inflexión en el que los escenarios de congestión resultan críticos para mantener la disponibilidad de los servicios. Ante esta necesidad, existen diferentes perspectivas que van, desde enfoques que abordan el comportamiento dinámico de la demanda de servicios, hasta el uso de técnicas de optimización de recursos usando técnicas de *machine learning*. Sin embargo, estas perspectivas suelen omitir la alta disponibilidad en la entrega de servicios, por lo cual, paradigmas como NFV y SDN facilitan la adición de componentes elásticos a la arquitectura, con el fin de poder autoescalar recursos bajo demanda y mantener así la disponibilidad de los servicios. Aunque los mecanismos de autoescalamiento son ampliamente conocidos en soluciones orientadas a computación en la nube, la mayoría de ellos se basan en técnicas reactivas que afectan la entrega del servicio en el periodo de tiempo comprendido entre la identificación de la necesidad de escalar y el instante en que el escalamiento culmina.

Dicho lo anterior, en esta investigación se implementa un modelo predictivo para la asignación elástica de recursos sobre un entorno NFV/SDN basado en herramientas de código abierto como OpenStack. Usando como referencia una arquitectura que puede implementarse en entornos de bajo costo mediante herramientas de código abierto, se adecúa una metodología de autoescalamiento basada en recomendaciones del 3GPP. Luego, utilizando el algoritmo HTM para predecir tendencias, se efectúan asignaciones proactivas de recursos según reglas de violación de umbral, definidas en un algoritmo de autoescalamiento que sintetiza la asignación elástica de recursos. Los datos que enriquecen el modelo predictivo se generan siguiendo la tendencia de la demanda de recursos de una red móvil real. Los resultados muestran que, a través del modelo propuesto, es posible reducir el tiempo entre identificar la necesidad de escalar y culminar el escalamiento, en comparación con soluciones conocidas de computación en la nube. Además, es posible mantener la disponibilidad del servicio mientras se mejora la latencia en el tiempo de conexión al mismo.

Palabras clave: Autoescalamiento; SDN; NFV; HTM; Infraestructura Virtual; Arquitectura de Código Abierto; Alta Disponibilidad; Modelo Predictivo.

Abstract

The evolution of the technologies used to maintain the delivery of quality services on telecommunications networks, has shown problems around the architectures with which it have been conceived. Mainly, the diversity in the type of connections and the massive delivery of services has brought the operation of the networks to a inflection point in which congestion scenarios are critical to maintain the availability of services. Faced with this need, there are different perspectives ranging from approaches that address the dynamic behavior of the demand for services, to the use of resource optimization techniques using machine learning techniques. However, these perspectives tend to omit high availability in service delivery, which is why paradigms such as NFV and SDN facilitate the addition of elastic components to the architecture, in order to be able to autoscaling resources on demand and thus maintain availability of the services. Although autoscaling mechanisms are widely known in cloud computing solutions, most of them are based on reactive techniques that affect the delivery of the service in the period of time between the identification of the need to scale and the moment where the escalation ends.

That said, this research implements a predictive model for the elastic allocation of resources on an NFV/SDN environment based on open source tools such as OpenStack. Using as a reference an architecture that can be implemented in low-cost environments using open source tools, an autoscaling methodology based on 3GPP recommendations is adapted. Then, using HTM algorithm to predict trends, proactive resource allocations are made based on threshold violation rules defined in an autoscaling algorithm that synthesizes elastic resource allocation. The data that enrich the predictive model is generated following the trend of the demand for resources of a real mobile network. The results show that, through the proposed model, it is possible to reduce the time between identifying the need to scale and completing the scaling compared to known cloud computing solutions. In addition, it is possible to maintain the availability of the service while improving the latency in connection time to it.

Keywords: Autoscaling; SDN; NFV; HTM; Virtual Infrastructure; Open Source Architecture; High availability; Predictive Model.

Esta tesis de maestría se sustentó el 02 de Junio de 2021 a las 2:00 pm,
y fue evaluada por los siguientes jurados:

Carlos Andrés Lozano Garzón, Ph.D.
Universidad de los Andes

Octavio José Salcedo Parra, Ph.D.
Departamento de Ingeniería de Sistemas e Industrial, Universidad Nacional de Colombia

Contenido

Lista de Figuras

Lista de Tablas

Lista de Abreviaturas

Abreviatura	Término
2G	Segunda generación de tecnologías de telefonía móvil
3G	Tercera generación de tecnologías de telefonía móvil
3GPP	3rd Generation Partnership Ptoject
4G	Cuarta generación de tecnologías de telefonía móvil
5G	Quinta generación de tecnologías de telefonía móvil
5GPP	5G Infrastructure Public Private Partnership
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
AWS	Amazon Web Service
BSS	Business support system
CLA	Close Loop Automation
CNF	Container Network Functions
CPU	Central Processing Unit
CSR	Cell-Site Router
DHCP	Dynamic Host Configuration Protocol
DNN	Deep Neural Network
DNS	Domain Name Server
E2E	End-to-End
ELM	Extreme Learning Machine
eNB	eNodeB
EPC	Evolved Packet Core
ESN	Echo State Networks
ETSI	European Telecommunications Standards Institute
GCP	Google Cloud Platform
GENI	Global Environment for Networking Innovations
GPU	Graphics Processing Unit
GRNN	Generalized Regression Neural Network
HPA	Horizontal Pod Autoscaling
HTM	Hierarchical Temporal Memory
HTTP	Hypertext Transfer Protocol

Abreviatura	**Término**
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
KPI	Key Performance Indicator
KVM	Kernel-based Virtual Machine
L2	Layer 2
LSTM	Long-Short Term Memory
LTE	Long Term Evolution
MANO	Management and Orchestration
MAPE	Mean Absolute Percent Error
MEC	Mobile Edge Computing
ML	Machine Learning
MLP	Multilayer Perceptron
MME	Mobility Management Entity
MOS	Mean Opinion Score
NFV	Network Function Virtualization
NFVI	Network Functions Virtualization Infrastructure
NIC	Network Interface Card
NMS	Network Management System
NSI	Network Slice Instance
NS-2	Network Simulator version 2
ONAP	Open Network Automation Platform
OS	Operative System
OSM	Open Source MANO
OSS	Operations Support Systems
PaaS	Platform as a Service
PDCCH	Physical Downlink Control Channel
QoE	Quality of Experience
QoS	Quality of Service
RALF	Resource Allocation based on Load Forecasting
RAM	Random Access Memory
RAN	Radio Access Network
RBF	Radio Base Function
REST	Representational State Transfer
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
S3P	Stability, Security, Scalability and Performance

Abreviatura	Término
SaaS	Software as a Service
SDN	Software Defined Networking
SDR	Sparse Distributed Representation
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SON	Self-Organizing Network
SP	Spatial Pooler
SSH	Secure Shell
TCP	Transmission Control Protocol
TM	Temporal Memory
TSDB	Time Series Database
VIM	Virtual Infrastructure Management
VM	Virtual Machine
VNF	Virtual Network Functions
WMAN	Wireless Metropolitan Area Network

1. Introducción

1.1. Motivación

El despliegue masivo de servicios y el creciente número de dispositivos conectados a las redes de telecomunicaciones, ha llevado a los operadores de red a un punto de inflexión en el que deben replantear los sistemas que soportan su operación y negocio [1]. Históricamente, los operadores han tomado ventaja de los avances tecnológicos en infraestructura de red, para mejorar características de sus arquitecturas, aumentando el desempeño percibido por los usuarios en cuanto a velocidad de descarga y de subida, latencia en la comunicación y disponibilidad del servicio [2]. La mejora de estas características conlleva a que los modelos de negocio se actualicen con el propósito de aprovechar los avances técnicos en las redes que soportan la oferta de servicios. Particularmente, en el despliegue masivo de las tecnologías 2G y 3G, esta oferta se centraba en los servicios de voz y datos, enfocando a los operadores hacia un modelo basado en la operación de la red. Sin embargo, a partir de la transición a 4G, la arquitectura se simplifica y permite orientar el modelo hacia la entrega de servicios de calidad. Así mismo, se espera que con el despliegue masivo de 5G, los operadores centren sus modelos de negocio en la experiencia percibida por los usuarios cuando demandan un servicio a través de su red [3].

En el escenario venidero que aborda el despliegue de 5G, los operadores enfrentan desafíos enmarcados por la tendencia a la conectividad de una amplia variedad de dispositivos ubicados en el dominio del usuario y a una nueva gama de servicios, cada uno con requisitos individuales de calidad. De este modo, se espera que para el año 2022 el tráfico que curse las redes móviles se duplique respecto al del año 2019. Además, se destaca que, de todo ese tráfico, el 63 % corresponde a servicios multimedia [4]. Esta tendencia al consumo, evidencia la necesidad de poseer infraestructuras de red que soporten la entrega de servicios considerando su ritmo de crecimiento. Por otro lado, hay que tener en cuenta que los usuarios esperan que el servicio se encuentre disponible la mayor parte del tiempo y que sea de alta calidad durante su consumo. Por este motivo, son varias las investigaciones enmarcadas en dinamizar, automatizar, escalar o proponer nuevos enfoques para la operación de las redes.

En las arquitecturas de red tradicionales, se tiene una dependencia en el tipo de dispositivo físico que ejecuta las funciones de red. Si bien este enfoque provee soluciones efectivas para conectar usuarios con los contenidos o servicios, no es adecuada para la gestión y orquestación

de la red, pues hace rígida la arquitectura [5]. En la Tabla **1-1** se describen de manera breve algunas investigaciones que potencian la conexión entre usuarios y servicios, pero carecen de capas de gestión u orquestación.

Tabla 1-1.: Investigaciones que potencian la conexión entre usuarios y servicios.

Enfoque	Problema	Solución	Aporte
Efectos de la contienda por acceso al medio	El consumo simultáneo de servicios genera una condición de asignación no eficiente por parte del protocolo TCP, que disminuye la eficiencia en el uso del ancho de banda del canal.	Se plantea un escenario con múltiples usuarios consumiendo de forma simultánea un servicio de *videostreaming*, pero haciendo uso de protocolos adaptativos.	Se demuestra la necesidad de una capa de gestión que administre los instantes de tiempo en los que cada usuario puede hacer uso del canal [6].
Optimización de rutas	Los servicios multimedia son priorizados en el enrutamiento sin considerar la percepción del usuario.	Se implementan algoritmos de enrutamiento que priorizan la entrega de paquetes según el servicio al que pertenece. Las soluciones se centran en el servicio de *videostreaming*.	Se evidencia que una red programable o con una capa para aplicaciones de red, permite priorizar el enrutamiento con base en métricas relacionadas a la percepción del usuario, como el MOS, o al desempeño de la red, como la QoS [7, 8, 9].
Arquitecturas híbridas	Con las arquitecturas tradicionales no es posible satisfacer la demanda de recursos de aplicaciones multimedia y servicios en línea con garantía de QoS.	Se opta por un enfoque donde se superponen SDN y una red tradicional. Desacoplando los planos de control y datos, se logra diferenciar los servicios a través de un *gateway* programable.	Se evidencia una transición de arquitecturas orientadas de dispositivos físicos, a arquitecturas que logran desacoplar las funciones del dispositivo que las ejecuta [10].

Un factor común encontrado en estas investigaciones es que las arquitecturas tradicionales suelen hacer un dimensionamiento de recursos de red basado en reglas estáticas. Sin embargo, el comportamiento dinámico del tráfico sugiere que las capacidades de la red deben ser dinámicas [11]. En [12], este comportamiento estático lo definen como "osificación"de la red. Por esta razón, si la demanda es más de lo que se esperaba, la red es vulnerable a escenarios de congestión. De igual forma, si es menor, se dice que hay sobredimensionamiento de recursos [13]. Ante estos inconvenientes se han realizado investigaciones en las que se optimiza el uso de recursos usando técnicas de ML (Machine Learning) para maximizar la QoS (Quality of Service) o la QoE (Quality of Experience). La Tabla **1-2** resume algunos trabajos que consideran este enfoque.

Tabla 1-2.: Investigaciones que usan técnicas de ML para mejorar métricas de calidad.

Enfoque	**Problema**	**Solución**	**Aporte**
QoS adaptativa para comunicaciones en tiempo real considerando la percepción del usuario cuando demanda un servicio.	Los criterios de enrutamiento estáticos pueden degradar la calidad de un servicio cuando se presentan fluctuaciones en el comportamiento del tráfico.	Se implementan algoritmos de ML para clasificar comportamientos del tráfico de red y priorizar el enrutamiento según este comportamiento. Se suele usar la latencia como criterio de mejora.	Se demuestra que la aplicación de técnicas de ML al enrutamiento, mejora la QoS percibida por los usuarios, agrega flexibilidad al establecimiento de conexiones con la red y hace más eficiente el uso de recursos de red [14, 15, 16].
Mejora de la QoE usando técnicas adaptativas aplicadas a servicios multimedia.	La percepción de los usuarios suele ser más sensible cuando consumen servicios multimedia, por lo que no se garantiza la calidad únicamente con métricas de red. Hay que considerar métricas de percepción del servicio obtenidas desde el dominio del usuario.	Se implementan algoritmos que priorizan los paquetes de servicios multimedia, con base en métricas de percepción del usuario. Se suele usar como criterio de mejora, la tasa de descarga del servicio o el MOS.	Se presenta una transición de la operación basada en el servicio, como suele hacerse en 4G, a una operación basada en el usuario, como espera hacerse en 5G [17, 18].

Aunque las técnicas de ML mejoran la eficiencia aprendiendo el comportamiento dinámico de la red, no garantizan la alta disponibilidad de un servicio puesto que son implementadas sobre recursos ya preestablecidos. Por esta razón, es necesario abordar las limitaciones que imponen las arquitecturas tradicionales en cuanto a rigidez. En consecuencia, paradigmas de virtualización de recursos como NFV (Network Function Virtualization) y SDN (Software Defined Networking), aportan a la arquitectura de red, la gestión dinámica y centralizada, la programabilidad de funciones, y un componente elástico en la asignación de recursos. Esto último, es de interés central en esta investigación. Una definición más amplia de NFV y SDN puede consultarse en el Anexo A.

La combinación de NFV y SDN permite la implementación de infraestructuras de red virtuales con ventajas como el uso eficiente de recursos, el monitoreo constante de su estado y la alta disponibilidad en el despliegue de servicios. Por otra parte, en este tipo de infraestructuras, los componentes elásticos se refieren a la capacidad de escalar recursos como CPU (Central Processing Unit), RAM (Random Access Memory), almacenamiento interno, interfaces de red, ancho de banda o instancias, bajo demanda. Por ende, está ampliamente relacionado con conceptos como autoescalamiento [19]. A manera de ejemplo, cuando se presentan aumentos de tráfico inminentes, los elementos de red pueden ser provistos con mayor RAM, CPU o recursos de red, de manera que el impacto de un posible evento de congestión se minimice reduciendo el tiempo en el que se afecta la calidad de los servicios. Así mismo, en horas de baja demanda como las madrugadas, los recursos pueden reducirse hasta el punto de mantener la disponibilidad de los servicios con una calidad idéntica a la ofrecida en horas de alta demanda; esto con el fin de reducir costos de operación. Es importante mencionar, que combinar la elasticidad con la gestión centralizada, permite automatizar la operación de la red al punto en el que no se requiere intervención humana. Este tipo de redes suelen denominarse SON (Self-Organizing Networks) [20]. Otras investigaciones las enmarcan bajo conceptos como CLA (Close Loop Automation) [21] o *Zero-Touch operation* [22]. Estas redes suelen tener un flujo de extremo a extremo (E2E, *end-to-end*), donde se monitorean los recursos de la red, se infiere su estado (subdimensionado, óptimo o sobredimensionado) y se toman acciones correctivas de asignación. En la Tabla **1-3** se presentan trabajos relacionados con autoescalamiento sobre infraestructuras de red virtuales.

1.2. Justificación

La implementación de mecanismos de autoescalamiento es un punto clave para el desarrollo de tecnologías asociadas a 5G dado que habilita la gestión autónoma de la infraestructura de red [23]. De hecho, existen recomendaciones del 3GPP (3rd Generation Partnership Project) que promueven la implementación de casos de uso relacionados a través de una descripción detallada de los pasos y requerimientos que deben considerar los operadores antes de considerar algún caso [20].

Tabla 1-3.: Investigaciones sobre autoescalamiento en arquitecturas NFV/SDN.

Enfoque	Problema	Solución	Aporte
Asignación de recursos con base en mediciones de QoS o QoE.	Cómo optimizar la calidad percibida por los usuarios en escenarios vulnerables a congestión, manipulando la asignación de recursos como el ancho de banda.	Se implementan algoritmos de ML para la creación adaptativa de rutas que maximicen la tasa de descarga percibida por los usuarios.	Usando únicamente el componente SDN de la arquitectura, es posible implementar enfoques de autoescalamiento de rutas para mejorar la calidad en la entrega de un servicio [24].
Predicción de *throughput* para anticipar la toma de acciones correctivas.	Esperar a que los aumentos de tráfico ocurran para tomar acciones correctivas puede traer periodos de congestión que afectan la calidad percibida por los usuarios.	Se implementan técnicas de ML que predicen el comportamiento del tráfico en ventanas de hasta varios días para anticipar posibles picos y tomar acciones preventivas. Se destaca que las investigaciones se enfocan en LTE.	Se demuestra que la aplicación de técnicas predictivas puede evitar periodos de afectación del servicio en redes móviles [25, 26, 27, 28].
Gestión de recursos de infraestructura desde una capa de orquestación.	La gestión centralizada de las infraestructuras de red virtuales, sugiere que puede usarse un componente externo para la administración de recursos. Sin embargo, es necesario considerar un flujo E2E que va desde el monitoreo de recursos, hasta efectuar tareas sobre los recursos.	Se implementa un *framework* para el aprovisionamiento de funciones virtuales de red que usa técnicas de ML para satisfacer requerimientos de demanda y evitar la violación de SLAs.	Se destaca el uso de herramientas de simulación para evidenciar la eficacia de la capa de orquestación en la administración de recursos. Además, se demuestra que implementar una capa de orquestación, facilita las tareas de gestión de recursos [29, 30].

Sin embargo, es común encontrar en los trabajos previos que los escenarios de experimentación sobre los cuales se implementan técnicas elásticas o mecanismos de autoescalamiento suelen carecer de infraestructuras de red o elementos reales. Así mismo, la mayoría de las investigaciones se centra en comportamientos reactivos de la red. En otras palabras, las técnicas de escalamiento revisadas son aplicadas cuando se presenta una anomalía en la demanda de recursos y no antes, añadiendo un periodo de afectación del servicio mientras la corrección por escalamiento de recursos es efectuada. Este comportamiento puede ser corregido si se hace uso de mecanismos proactivos, como las que están basados en predicción de recursos. Mediante estos, es posible disminuir el periodo de afectación del servicio, puesto que la posible afectación es detectada antes que ocurra, dando un periodo de guarda para tomar las acciones correctivas y ejecutar el escalamiento de recursos necesario [29].

1.3. Identificación del problema

Como se discutió, se han llevado a cabo investigaciones que predicen requerimientos de recursos en infraestructuras de red desde diferentes enfoques, usando técnicas de ML para clasificar características que infieran la necesidad de recursos, o aprovisionando recursos virtuales según la predicción de su demanda. Este tipo de enfoques resulta útil para reducir los tiempos de acciones correctivas sobre la infraestructura.

Aunque los tiempos de escalamiento u optimización de recursos se pueden reducir a través de predicciones, los ambientes en los que se implementan suelen carecer de componentes que permitan acercarse al comportamiento que podría presentarse en una infraestructura de red real. Soluciones populares de autoescalamiento en gestores de infraestructura como GCP (Google Cloud Platform), garantizan que se requiere al menos 10 minutos para que, una vez detectado el requerimiento de escalamiento, se estabilice el funcionamiento de la infraestructura [31]. Para tener una mayor flexibilidad en la configuración y operación de la infraestructura de red que soporta el escenario de experimentación controlado, esta investigación contempla el uso de una VIM (Virtual Infrastructure Management) de código abierto como OpenStack [32], puesto que ha sido ampliamente desplegada para la gestión de infraestructuras virtuales y sobre la cual se han construido varias distribuciones (FusionSphere de Huawei, WhiteStack, DevStack, MicroStack, etc.), permitiendo un acercamiento a un escenario real de operación de las redes virtualizadas. Dicho esto, surge la siguiente pregunta de investigación: ¿cómo implementar un modelo predictivo de asignación elástica de recursos usando técnicas de *machine learning*, como redes neuronales, sobre entornos NFV/SDN basados en ambientes reales de infraestructura de red como OpenStack?

1.4. Objetivo general y objetivos específicos

1.4.1. Objetivo general

Implementar un modelo predictivo usando redes neuronales para la asignación elástica de recursos sobre entornos NFV/SDN basados en OpenStack.

1.4.2. Objetivos específicos

1. Desarrollar un algoritmo de predicción de series de tiempo basado en redes neuronales que estime la demanda de recursos para entornos NFV/SDN.

2. Especificar las políticas de autoescalamiento de la red NFV/SDN con base en mediciones históricas del uso de recursos obtenidas a partir del monitoreo de la red en OpenStack.

3. Ejecutar estrategias correctivas sobre los recursos de la infraestructura de red siguiendo recomendaciones SON para un entorno NFV/SDN basado en OpenStack.

1.5. Metodología

El 3GPP, en su estudio para redes 5G, ha levantado una serie de casos de uso que usan SON como característica esencial de funcionamiento [20]. Entre ellos, se encuentra la optimización transversal de recursos de red, de interés en esta investigación. El objetivo de este caso de uso es optimizar la asignación de recursos realizada a múltiples NSI (Network Slice Instance), considerando el total de recursos físicos o virtuales disponibles. Para implementarlo, se hace una adaptación de los pasos propuestos dando como resultado la metodología propuesta en la Figura **1-1**.

Los pasos resaltados en negrilla en la Figura **1-1** corresponden a adaptaciones realizadas en el contexto de la propuesta. Por otra parte, dado que la experimentación se realizará sobre una infraestructura de red emulada en un ambiente controlado sobre OpenStack, la investigación es de carácter cuantitativo. Cada paso de la metodología se describe como sigue:

1. **Definir la red:** Como paso inicial, se considera la implementación de una topología de red en donde existan tres elementos clave: 1) herramientas para extracción y almacenamiento de métricas, 2) un gestor de infraestructura virtual y 3) un elemento en una capa superior que haga las veces de gestor de red. Por lo tanto, es necesario desplegar un ambiente de red NFV/SDN vulnerable a escenarios de congestión y sobredimensionamiento de recursos usando OpenStack.

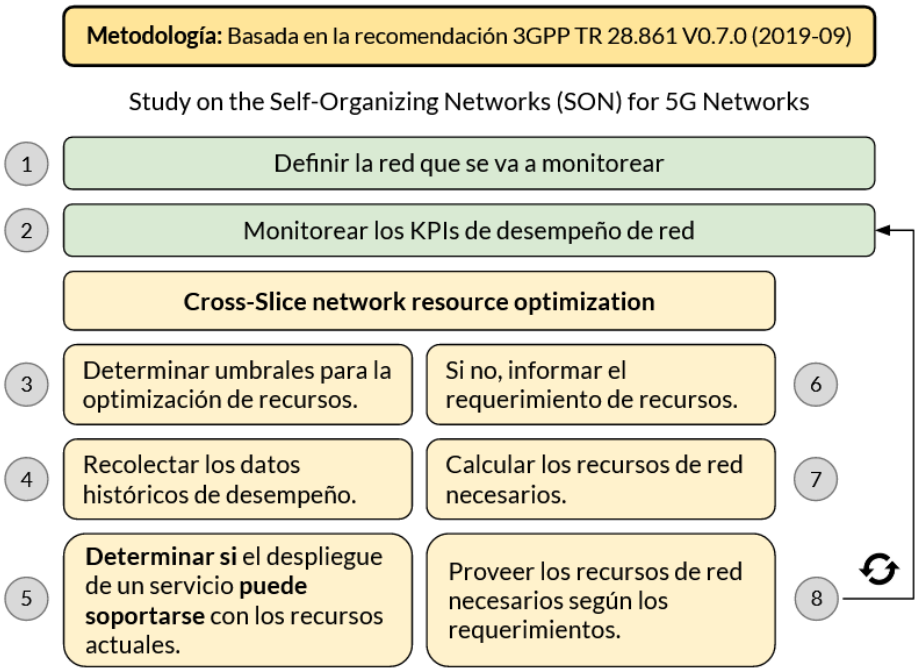


Figura 1-1.: Metodología para el desarrollo de la investigación. Adaptada de [20].

2. **Monitorear recursos:** Con la red implementada, se extraen métricas de la VIM asociadas al consumo de recursos. Las mediciones obtenidas van hacia una capa de gestión para su monitoreo y control. Luego, es necesario definir las métricas que permitan monitorear los recursos de interés en escenarios de congestión y sobredimensionamiento.

3. **Definir umbrales:** Con base en el comportamiento normal de las métricas definidas, se establecen los umbrales para activar la optimización de recursos, es decir, aumentar o disminuir capacidades en las VMs (Virtual Machine) desplegadas, ancho de banda de las redes internas o número de interfaces.

4. **Recolectar datos:** Se recolectan datos históricos de las métricas consideradas en el desarrollo de la investigación.

5. **Determinar condiciones para el despliegue de un servicio:** Mediante un algoritmo de aprendizaje automático para predicción de tendencias, basado en redes neuronales, se predice el uso de recursos usando como entrada los datos recolectados de las métricas claves.

6. **Informar requerimientos:** Si la predicción de recursos arroja requerimientos que no se ajustan con los recursos provistos con la infraestructura actual, se informa a la capa de gestión su falta o exceso.

7. **Calcular los recursos necesarios:** Estimar cuantitativamente con cuántos recursos de infraestructura se satisfacen los requerimientos predichos.

8. **Proveer recursos de red:** Asignar los recursos a la infraestructura. Una vez se ajustan los recursos, se da paso nuevamente a su monitoreo.

1.6. Estructura del documento

Este documento de tesis de maestría está organizado de tal manera que cada capítulo representa una investigación independiente enmarcada dentro del objetivo general. El Capítulo 2 presenta los bases técnicas necesarias para el desarrollo de la investigación. Entre tanto, los objetivos específicos son alcanzados en los Capítulos 3 y 4. En detalle:

- En el Capítulo 2 se presenta una revisión de cuáles han sido las investigaciones que han implementado escenarios asequibles para el despliegue de mecanismos de autoescalamiento. Durante le revisión, se priorizan las investigaciones que usan OpenStack como infraestructura de despliegue. Además, en este mismo capítulo, se propone una arquitectura de bajo nivel para el despliegue de un escenario asequible basado en herramientas de código abierto. Este capítulo es transversal a todos los objetivos específicos y representa la base necesaria para el desarrollo técnico de la investigación.

- En el Capítulo 3 se presenta una revisión de las investigaciones más relevantes en cuanto al uso de técnicas de predicción para resolver problemas asociados a redes de telecomunicaciones. Durante la revisión, se priorizan las investigaciones que consideren el algoritmo de predicción HTM (Hierarchical Temporal Memory) en sus experimentos. Además, se presenta un marco conceptual para entender el funcionamiento interno de HTM. En este capítulo se presenta una implementación de este algoritmo con el propósito de predecir el comportamiento de una serie de tiempo de prueba, construida a partir de información de tráfico de una red móvil real. Este capítulo aborda el primer objetivo específico.

- En el Capítulo 4 se presenta una revisión de las investigaciones enfocadas a implementar mecanismos de autoescalamiento en elementos de una red de telecomunicaciones. En la revisión se priorizan investigaciones enfocadas en el escalamiento de recursos sobre infraestructuras de red virtuales. En este capítulo se presenta un mecanismo de autoescalamiento usando Terraform, que considera las predicciones de HTM para efectuar asignaciones preventivas de recursos sobre una infraestructura basada en OpenStack. El escalamiento es definido mediante políticas estáticas basadas en reglas de violación de umbral. Este capítulo aborda el segundo y el tercer objetivo específico.

- En el Capítulo 5 se resumen las principales conclusiones obtenidas en la realización de esta investigación y se proponen trabajos futuros.

2. Escenario de experimentación basado en OpenStack para la ejecución de políticas de autoescalamiento

2.1. Introducción

El volumen de información que transportan hoy en día las redes de telecomunicaciones, es del orden de las decenas de exabytes si se mide mensualmente [33]. Todo tipo de información es transportada, desde mensajes de texto hasta contenido multimedia, siendo este último el más significativo con más del 66 % del total transportado. Por otra parte, en Latinoamérica, la cantidad de tráfico generado por dispositivo, en redes móviles, es cercano a 6 GB por mes y se espera que al 2026 la cifra ascienda a 26 GB por mes. A su vez, de la cantidad de suscriptores de redes de acceso móvil, el 59 % se conecta mediante LTE (Long Term Evolution). Para el 2026 se espera que esa cifra descienda a un 56 %, dando paso a que el 27 % de los suscriptores se conecten mediante tecnologías 5G, con un incremento progresivo de la cantidad de dispositivos conectados a la red [4]. El advenimiento de 5G trae consigo ventajas tecnológicas que permiten el despliegue y consumo de servicios con alta demanda de ancho de banda, como la transmisión de contenidos multimedia de ultra alta definición o, servicios que necesitan una alta confiabilidad en la comunicación, como la telemedicina.

Estas condiciones impuestas por el despliegue masivo de servicios, las expectativas de los clientes y los nuevos casos de uso propuestos han llevado a las arquitecturas de red a un punto de inflexión donde deben soportar los servicios sin degradar la experiencia percibida por los usuarios, además de considerar el crecimiento progresivo de la demanda. Bajo este escenario, las arquitecturas de red tradicionales pasan a un segundo plano de despliegue debido a su poca flexibilidad y escalabilidad, causada principalmente por la dependencia hacia el tipo de hardware usado y su proveedor. Por esta razón, los operadores de red han venido adoptando soluciones basadas en arquitecturas virtualizadas para el despliegue de sus elementos de red. Con este nuevo enfoque, los operadores logran materializar el concepto de despliegue/repliegue bajo demanda, habilitando modelos como SON [20], CLA [21] y *Zero-Touch operation* [22]. Además, desacoplan el hardware de sus funciones, permitiendo una apertura entre el plano de datos y el plano de control en la red. Con una red abierta, los operadores pueden añadir aplicaciones personalizadas, que pueden ir desde técnicas de

asignación de recursos basadas en reglas de violación de umbral, hasta aplicaciones de ML enmarcadas en automatización de procesos [34].

Las arquitecturas virtualizadas suelen ser implementadas usando nubes públicas o privadas, adoptando conceptos de computación en la nube y enfoques basados en contenedores [35]. A través de estos conceptos, los recursos de cómputo, como CPU, RAM, recursos de almacenamiento y recursos de red, pueden escalarse dinámicamente bajo demanda o de forma proactiva. Cualquier tipo de aplicación desarrollada bajo esta metodología tiene el potencial de escalar (aumentar/descender) los recursos virtuales utilizados para alcanzar alta disponibilidad, alto desempeño y eficiencia. Esta habilidad de escalamiento (horizontal o vertical), es una de las mayores ventajas de la computación en la nube [36]. Sin embargo, la implementación de un escenario de este tipo, puede necesitar múltiples servidores físicos desplegados en un centro de datos *on-premise* o suscripciones a nubes públicas como GCP, AWS (Amazon Web Services) o Microsoft Azure.

En este capítulo se describe la implementación de un escenario de computación en la nube de bajo costo, asequible y local, útil para la ejecución de experimentos basados en la asignación dinámica de recursos virtuales o implementación de mecanismos de autoescalamiento. El escenario está basado en OpenStack [30]. Además, considera la implementación de los contenedores de Prometheus [37] y Grafana [38] para la extracción, almacenamiento y monitoreo de métricas asociadas a los recursos de cómputo, almacenamiento y red. Por último, contempla la implementación de una capa de orquestación, en la que se usa Terraform [39] como agente externo para la ejecución de políticas de autoescalamiento y algoritmos de predicción de demanda de recursos que añaden la inteligencia a la arquitectura propuesta para determinar cuándo ejecutar la asignación de recursos.

2.2. Trabajos relacionados que describen el escenario de experimentación para la implementación de mecanismos de escalamiento

La adopción de tecnologías basadas en computación en la nube, ha provocado que muchas aplicaciones sean migradas hacia ese tipo de ambiente. Una vez allí, se pueden explotar características como asignación elástica de recursos, autoescalamiento, alta disponibilidad en los servicios desplegados y disminución en los costos debido al uso eficiente de recursos [40]. Sin embargo, disponer de un escenario de computación en la nube local, asequible y para propósitos experimentales, no resulta sencillo. Es común encontrar el uso de nubes públicas como AWS, GCP o Microsoft Azure para experimentos relacionados con asignación de recursos o evaluación de desempeño en el despliegue de aplicaciones. No obstante, el uso de estas nubes acarrea un costo *pay-as-you-go* por cualquier recurso usado. Por esta razón,

en el contexto de esta investigación, se prioriza la revisión de investigaciones que hacen uso de OpenStack. Adicionalmente, se resaltan investigaciones que experimentan técnicas de autoescalamiento o asignación dinámica-elástica de recursos. La revisión también se enfoca en qué hardware base fue usado para la experimentación, pues es útil como punto de comparación.

Cuando se adopta una solución de computación en la nube, se dispone de un catálogo de servicios que busca usarse en pro del objetivo para el cual ha sido pensada. Cualquier solución de nube pública o privada provee, por ejemplo, mecanismos de autoescalamiento. Sin embargo, estos servicios están atados al proveedor de nube, por lo que muchos autores han decidido implementar sus propios mecanismos. En [41] los autores proponen un servicio que escala aplicaciones de forma proactiva y dinámica según reglas de desempeño establecidas por los usuarios. Una vez definidas las reglas, son garantizadas mediante una técnica de control basada en filtros de Kalman. La autonomía otorgada al sistema, reduce hasta en un 25 % el consumo de recursos en comparación con otras soluciones. El servicio es implementado sobre OpenStack en un ambiente de experimentación de múltiples hipervisores de 8 CPU y 8 GB de memoria cada uno. A pesar de usar hipervisores relativamente livianos, posee inconvenientes como: la arquitectura propuesta por los autores, infiere el uso de múltiples ambientes de computación en la nube y el servicio desarrollado requiere uno de esos ambientes para ejecutarse y actuar como agente externo, expandiendo la arquitectura y aumentando los costos de despliegue. Este inconveniente de sobrecostos es común. Por esta razón, la investigación realizada en [42], discute acerca de la ejecución paralela de procesos de cómputo, como metaheurísticas sobre infraestructuras virtuales para resolver problemas de optimización del mundo real a precios asequibles. Suele ocurrir que el costo computacional para la ejecución de las metaheurísticas, limita a las instituciones que investigan su uso. Por consiguiente, en dicha investigación se propuso un framework denominado WoBinGo, el cual optimiza procesos mediante computación en paralelo sobre soluciones de nubes públicas como AWS o nubes privadas como OpenStack. Aunque es claro el método, los autores no hacen explícitos los recursos de hardware usados para la experimentación.

Otras investigaciones consideran el uso de técnicas predictivas para autoescalar recursos y minimizar su uso, sin desmejorar la calidad. En [36] se usa OpenStack, integrado con InfluxDB y Grafana, para exportar métricas hacia un proceso de predicción que dispara la asignación de recursos de forma proactiva mediante OpenStack Heat. Aunque el escenario tiene una arquitectura con aplicaciones de baja demanda de recursos, recae en el uso de un sistema de escalamiento inmerso en OpenStack, lo que rigidiza su arquitectura, pues se necesitan tantos Heat como OpenStack usados. Existen otros escenarios que aprovechan otros módulos de OpenStack para mejorar el desempeño. Por ejemplo, en [43] se presenta una solución práctica para acelerar el desempeño de red de aplicaciones que demandan

una alta transferencia de datos. La solución ayuda a aprovisionar automáticamente redes *multi-tenant* que explotan las capacidades del hardware existente (por ejemplo, NIC, GPU y conmutadores SDN) sin necesidad de usar aceleradores de hardware adicionales. Los resultados muestran que el hardware puede llegar a su límite práctico de recursos de red para asegurar desempeño y confiabilidad en este tipo de aplicaciones.

Un inconveniente importante en el contexto del autoescalamiento, es la capacidad para definir y usar un plan de reserva de recursos, pues autoescalar no debe implicar ineficiencia en su uso. La planeación suele ser realizada por un administrador de infraestructura bajo dos posibles casos: reserva para uso inmediato y reserva para uso futuro. En [44] se propone un método en el que se considera el segundo escenario. Los autores diseñan un sistema para reserva adaptativa de recursos, usando técnicas de ML. El sistema itera sobre los resultados de planes previamente definidos, con el fin de obtener los insumos necesarios para crear los modelos de demanda de recursos. Los modelos son entonces reaplicados sobre la infraestructura, basada en OpenStack, y a su vez enriquecen nuevas salidas. Los resultados muestran que con este enfoque se mejora la utilización de recursos. Por otro lado, los requerimientos de hardware usados, son una única máquina virtual con 4 CPU, 16 GB de memoria y 100 GB de disco, lo cual demuestra que se pueden ejecutar experimentos de autoescalamiento sobre recursos asequibles.

Por otra parte, se han desarrollado aplicaciones que tienen una demanda asequible de recursos de cómputo. En [45] se exploran las tecnologías de virtualización basadas en contenedores e hipervisores para desplegar un escenario de experimentación sobre computadores de 2-4 CPU, 4-8 GB de RAM y de esta manera, evaluar qué tecnología es mejor candidata desde la perspectiva de enfoques de autoescalamiento. La experimentación se enmarca en el contexto de MEC (Mobile Edge Computing), y los criterios de comparación son la latencia en el acceso radio y el ancho de banda requerido para acceder al contenido. La latencia es una métrica clave; para el 2013, se informaba que cada 100 ms de latencia le costaba a Amazon un 1 % en ventas [46]. Por esta razón, es común encontrar investigaciones enfocadas en minimizar esta métrica. Por ejemplo, en [47] los autores proponen *Natif*, un esquema de distribución de VNFs (Virtual Network Function) implementado sobre OpenStack, que mitiga la latencia de extremo a extremo sin desmejorar el *throughput* de la red. El esquema considera la correlación entre tráfico y CPU de las VNFs para reducir la interferencia durante la instanciación y así mejorar la latencia de red. Los resultados muestran que se mejora hasta un 188 % promedio comparado con otros métodos.

Hay otros enfoques que añaden una capa de orquestación para optimizar el uso de recursos y poder construir escenarios más livianos. Por ejemplo, usando Docker [48] y Kubernetes [49], los autores en [50] demuestran que es posible construir una aplicación nativa para la nube con una capa de orquestación. La raíz del trabajo yace en el uso de Kubernetes

HPA (Horizontal Pod Autoscaling), que ofrece un enfoque reactivo basado en umbrales para ajustar automáticamente los recursos requeridos según la demanda. Otro escenario, diseñado en una capa de orquestación, es propuesto en [51]. En este caso, los autores proponen un framework para gestionar múltiples tecnologías de bases de datos desplegadas en un ambiente de nube privada basado en OpenStack. Los recursos de hardware utilizados son 3 servidores *bare-metal* cada uno con 56 CPU, 250 GB de memoria y 1 TB de disco. La intención de los autores con el framework, es proveer un mecanismo de extremo a extremo para modelar operaciones de bases de datos de una forma autoorganizada. Aunque los autores presentan la arquitectura de la solución de forma detallada, no es claro cuáles deben ser los pasos para implementar un escenario de experimentación idéntico.

Las investigaciones revisadas muestran una tendencia en la que los enfoques de autoescalamiento son diseñados como un componente aparte de OpenStack, para usar exclusivamente este último como proveedor de infraestructura, análogo a un IaaS (Infrastructure as a Service). Esto hace que los escenarios se dimensionen en pro de los servicios o aplicaciones a desplegar y se definan las reglas de escalamiento con base en la calidad de experiencia o servicio que quiere entregarse al usuario. Sin embargo, no es claro cuál debe ser el punto de partida para implementar un escenario con estas características. Aunque los autores suelen mencionar la cantidad de recursos de cómputo límite en términos de CPU, RAM, disco y red, dan por hecho la disponibilidad de la infraestructura usada y no se exponen de manera suficiente las explicaciones técnicas sobre versiones de software, tipos de despliegue o integración entre aplicaciones. Por esta razón, en este capítulo se describe la implementación de un escenario de experimentación de bajo costo, asequible y local, útil para el desarrollo de conceptos sobre SON, CLA o *Zero-Touch operation*.

2.3. Escenario de experimentación propuesto

La Figura **2-1** muestra la arquitectura de bajo nivel del escenario propuesto para la experimentación en técnicas predictivas de autoescalamiento sobre recursos virtualizados. En la figura se aprecia el uso de dos *hosts*: uno que soporta la infraestructura virtual y otro que gestiona los recursos, añade la inteligencia predictiva y contiene las políticas de autoescalamiento.

Esta propuesta se basa en la arquitectura NFV sugerida por la ETSI (European Telecommunications Standards Institute) [52]. El escenario gira en torno a OpenStack, quien hace las veces de VIM. De OpenStack se toman las métricas que monitorean los recursos virtuales y, también sobre OpenStack, se aplican las políticas de autoescalamiento. El componente que añade la inteligencia de predicción necesaria en el enfoque proactivo, está soportado en algoritmos de inteligencia artificial basados en redes neuronales. El algoritmo de predicción usado en este trabajo se describe en el Capítulo 3, mientras que la capa de gestión de recursos

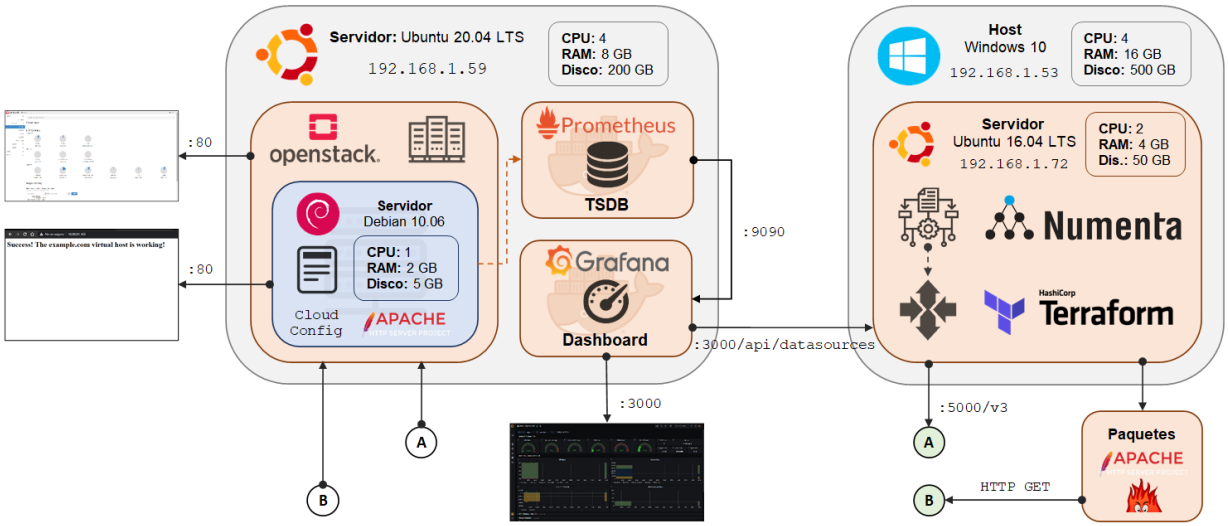


Figura 2-1.: Diseño de bajo nivel del escenario de experimentación propuesto.

se detalla en el Capítulo 4. En los siguientes apartados se describen los utilitarios usados en la arquitectura propuesta.

2.3.1. OpenStack

OpenStack es una plataforma de código abierto para computación en la nube, diseñada para la gestión de recursos en nubes públicas o privadas. A través de esta, se pueden administrar recursos virtuales traducidos en capacidad de cómputo (CPU y memoria), almacenamiento y red [32]. OpenStack está soportado por toda una comunidad de desarrolladores y usuarios finales, con nuevas versiones cada seis meses donde se incluyen mejoras, correcciones a problemas encontrados y características adicionales. Además, posee una excelente documentación y es ampliamente usado por diferentes operadores en el mundo como AT&T, Telefónica y Verizon. No obstante, la mayor ventaja es que puede correr sobre casi cualquier hardware estándar de propósito general, evitando la dependencia a proveedores específicos. La arquitectura de OpenStack se muestra en la Figura **2-2** y, como se observa en la figura, está compuesta de varios servicios que pueden desplegarse de manera independiente, permitiendo diferentes combinaciones al momento de desplegar soluciones personalizadas para la nube [53]. A continuación, se describen los principales servicios de OpenStack que son de interés para este trabajo.

OpenStack Compute (Nova)

El servicio que se encarga de administrar los recursos de cómputo se denomina *Nova*. Es el componente principal de OpenStack y permite el aprovisionamiento y control de los recursos virtuales que componen el sistema de cómputo en general. Aunque *Nova* puede usarse como

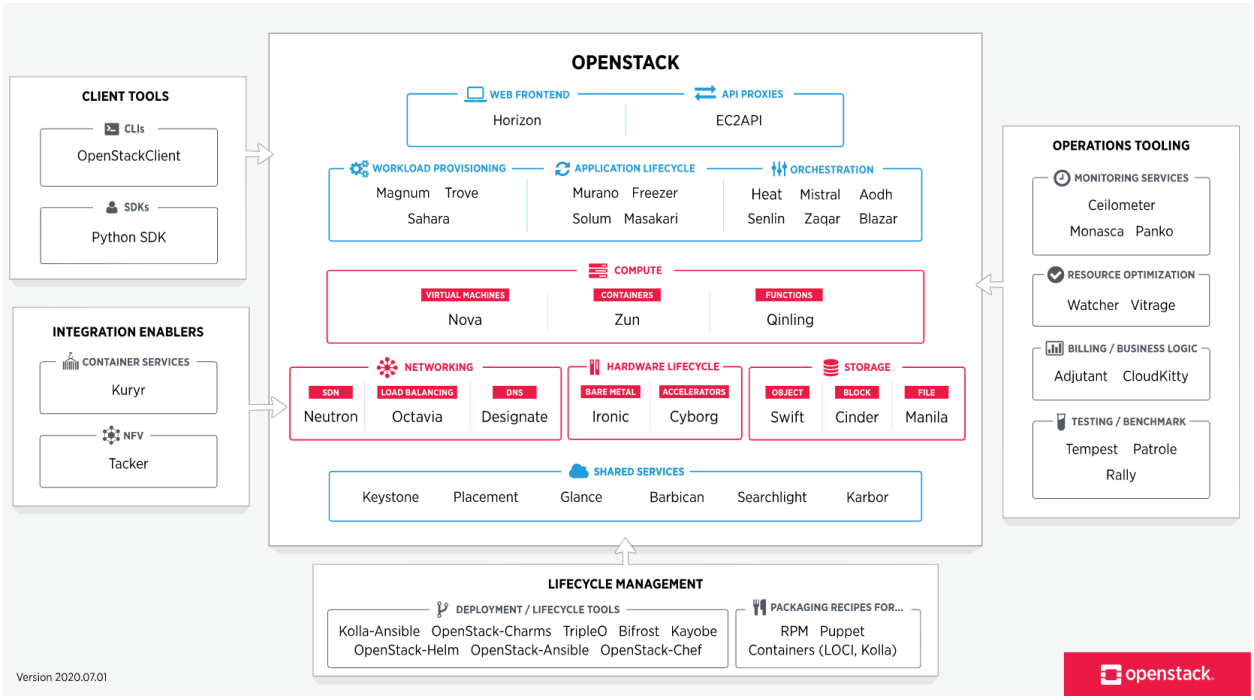


Figura 2-2.: Arquitectura de OpenStack (versión Ussuri) [32].

servicio independiente, en la arquitectura expuesta en la Figura **2-2** es funcional sólo si se comunica con los servicios de autenticación y autorización, almacenamiento de imágenes, *networking*, auditoría de recursos asignados y estado actual de los mismos.

OpenStack Image (Glance)

El servicio usado para registrar, descubrir, aprovisionar y almacenar imágenes de máquinas virtuales se denomina *Glance*. En la arquitectura de OpenStack, *Glance* provee una API REST a través de la cual también se pueden consultar otros datos relacionados a la VM o metadatos de la imagen usada. Los metadatos pueden incluir información como disco o volúmenes usados y/o formatos para habilitar virtualización a través de VMs internas o contenedores.

OpenStack Networking (Neutron)

El servicio usado para *networking* se denomina *Neutron*. Con este, se pueden crear y gestionar redes y/o subredes. Durante la creación de una instancia, *Neutron* se comunica con *Nova* para crear las interfaces de red y proveer la conectividad necesaria a las instancias. El modelo de *Neutron* incluye dos opciones para networking: redes pre-creadas o redes por *tenant*. Independientemente del modelo, se provee una conectividad L2 entre las instancias creadas y la infraestructura física. Luego, las instancias pueden tener una conectividad hacia redes externas a través de routers virtuales creados también con *Neutron*. Este mismo principio

aplica para la conectividad entre *tenants* o entre un *tenant* y la infraestructura física. Es importante aclarar, que las redes *tenant* son más sencillas de crear, pues no requieren un conocimiento profundo de la infraestructura de red física sobre la que se soporta OpenStack.

OpenStack Identity API (Keystone)

El servicio que provee una API para la autenticación, descubrimiento de servicios y autorizaciones *multi-tenant* se denomina *Keystone*. Con este servicio se generan llaves de autenticación que permiten acceder a los servicios de la API REST de OpenStack. Los clientes obtienen esta llave y la URL de los endpoints para que sus llamados a la API tengan credenciales válidas al momento de autenticarse.

Existen otros servicios en OpenStack que resultan útiles para los usuarios finales. Por ejemplo, el servicio de almacenamiento en bloque denominado *Cinder*, permite proveer volúmenes de grandes capacidades a las máquinas virtuales o contenedores desplegados. En la Figura **2-3** se muestra la interfaz gráfica que provee *Horizon*, diseñada para facilitar la administración de la infraestructura y el monitoreo de recursos. Es importante mencionar que OpenStack abstrae en una arquitectura virtual los recursos de una infraestructura física real. Al ser una solución de código abierto con el respaldo de toda una comunidad de desarrolladores y usuarios, además de su amplio uso por grandes operadores de red en el mundo, resulta adecuada para el escenario de experimentación propuesto.

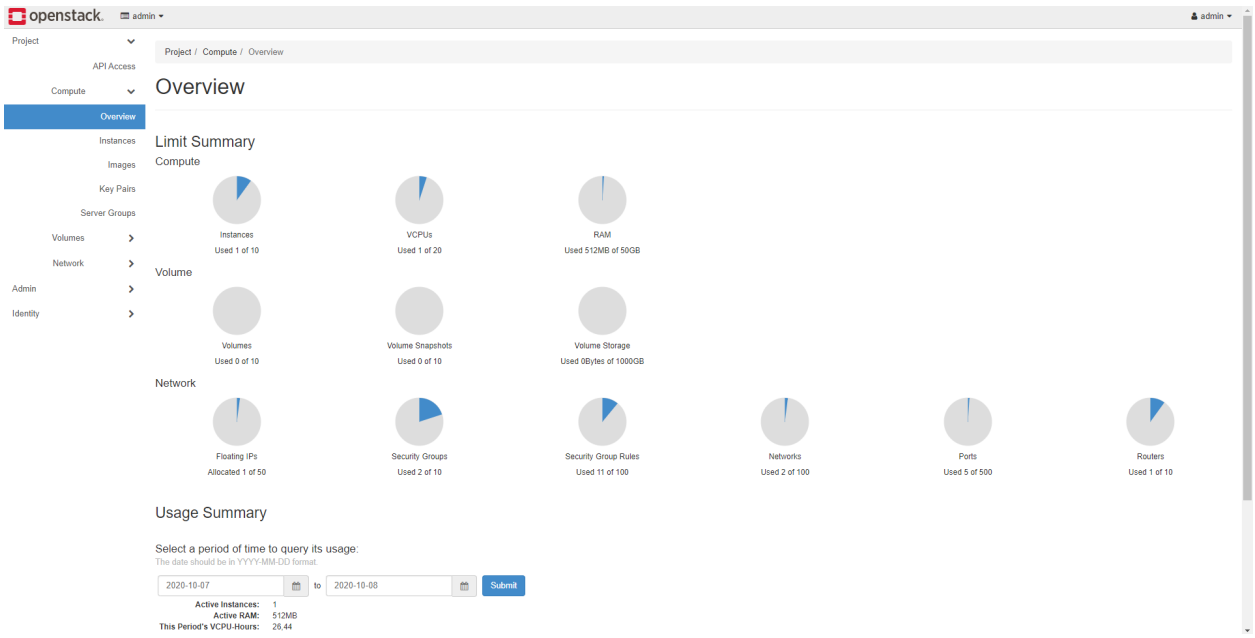


Figura 2-3.: Interfaz gráfica para la gestión de OpenStack.

Despliegue de OpenStack

Hay múltiples formas de desplegar OpenStack dependiendo del sistema operativo, de la cantidad de recursos, del tipo de despliegue (*on-premise*, *on-cloud*, etc.) o de la distribución. Esta última hace referencia a ramas que son desarrolladas por terceros (típicamente comerciales) para potenciar funcionalidades de OpenStack (por ejemplo, FusionSphere, Red Hat OpenStack Platform, WhiteStack WhiteCloud, etc.). La lista completa de distribuciones puede consultarse en el *marketplace* de OpenStack. Sin embargo, la mayoría están pensadas para ambientes productivos y demandan una gran cantidad de recursos para su despliegue. Por esta razón, en esta investigación se opta por hacer uso de MicroStack [54]. El despliegue de MicroStack puede consultarse en el Anexo B.

Extracción de métricas de OpenStack

Exportar y almacenar métricas desde OpenStack, puede hacerse de varias formas. La más popular es usando Gnocchi y Ceilometer [55]. No obstante, esto implica integrar dos aplicaciones adicionales al Stack de OpenStack. Por esta razón, en este trabajo se opta por una forma simplificada que consiste en definir, en un archivo *cloud init*, la instalación de un paquete de exportación de métricas hacia Prometheus. En el archivo, también se define el despliegue de un servidor Web, basado en *apache2*, destinado a pruebas de estrés y la instalación de un paquete para estresar RAM y CPU. Los archivos de *cloud init* son usados como receta para la creación de infraestructura en nube y son estándar para cualquier tipo de plataforma. Pueden ser usados en KVM, Vagrant, Virtualbox, nubes públicas como GCP, AWS o Azure y, por supuesto, OpenStack. A continuación, se adjunta el archivo de *cloud init* usado.

```
#cloud-config
package_upgrade: true

password: orion
chpasswd: { expire: False }
ssh_pwauth: True

runcmd:
 - [ sh, -c, 'echo "nameserver 8.8.8.8" >> /etc/resolv.conf' ]
 - sudo apt update && sudo apt install -y apache2
 - sudo systemctl restart apache2
 - sudo apt install -y prometheus-node-exporter
 - sudo apt install -y stress-ng
 - sudo mkdir -p /var/www/example.com
 - sudo chown -R $USER:$USER /var/www/example.com
 - sudo chmod -R 755 /var/www/example.com
```

```
17      - sudo echo -e "<html>\n\t<head>\n\t\t<title>Welcome to example.com\!</title
    >\n\t</head>\n\t<body>\n\t\t<h1>Success\!  The example.com virtual host is
    working!</h1>\n\t</body>\n</html>" >> /var/www/example.com/index.html
18      - sudo sed -i 's/\\//g' /var/www/example.com/html/index.html
19      - sudo echo -e "<VirtualHost *:80>\n\tServerAdmin webmaster@localhost\n\
    tServerName example.com\n\tServerAlias www.example.com\n\tDocumentRoot /var/
    www/example.com\n\tErrorLog /var/log/apache2/error.log\n\tCustomLog /var/log/
    apache2/access.log combined\n</VirtualHost>" >> /etc/apache2/sites-available/
    example.com.conf
20      - sudo a2ensite example.com.conf
21      - sudo a2dissite 000-default.conf
22      - sudo apache2ctl configtest
23      - sudo systemctl restart apache2
```

Los paquetes inicializados en cada máquina virtual permiten exportar métricas de consumo de recursos virtuales a Prometheus cada 30 segundos, disponibilizar un servidor Web y hacer pruebas de estrés. En Prometheus la información se almacena como series de tiempo independientes, con un histórico máximo por defecto de 5 días donde se habilita su agregación temporal y exposición a través de dos vías: la interfaz gráfica que se despliega con la aplicación o la API para consultas externas. En la siguiente sección, se detalla el monitoreo de estas métricas usando Prometheus y Grafana.

2.3.2. Monitoreo de recursos: Prometheus y Grafana

La combinación Prometheus y Grafana es ampliamente conocida y desplegada en ambientes productivos [56]. Prometheus es una herramienta de código abierto diseñada para monitorear, almacenar y alarmar datos en forma de secuencias. Usualmente, suele usarse como base de datos orientada a series de tiempo (TSDB, Time Series Database). Las principales características de esta herramienta son: permite el modelamiento de datos multi dimensionales, es independiente de almacenamientos distribuidos y tiene su propio lenguaje de consultas, el cual es flexible para soportar consultas de alta dimensionalidad [37]. Por otra parte, Grafana es una aplicación de *front end* de código abierto usada para la construcción de *dashboards* de monitoreo de métricas, principalmente en tiempo real. Permite consultar, visualizar, alarmar y explorar métricas de casi cualquier aplicación de base de datos [38]. Ambas aplicaciones tienen paquetes que pueden ser desplegados en casi cualquier sistema operativo y una documentación ampliamente constituida.

En el contexto del escenario propuesto, la principal ventaja de ambos, Prometheus y Grafana, es que tienen una versión contenerizada que puede ser desplegada usando Docker [48], facilitando la instalación e integración de ambas aplicaciones en la arquitectura propuesta. Como se observa en la Figura **2-1**, ambos contenedores son desplegados en

el mismo servidor donde se encuentra OpenStack (desplegado usando MicroStack). Se puede pensar que se necesitan recursos adicionales para su despliegue, pero al tratarse de tecnologías de virtualización, el servidor administra sus recursos, asignándolos con base en la exigencia de cada aplicación y manteniendo un límite para evitar la inestabilidad del *host* que las hospeda. Esta es una de las múltiples ventajas que trae el uso de tecnologías de virtualización de recursos. El despliegue de Prometheus y Grafana puede consultarse en el Anexo C. Una vez se configuran ambas herramientas, el resultado es un *dashboard* que muestra métricas relacionadas con el consumo de recursos, como el que se muestra en la Figura **2-4**.

Figura 2-4.: *Dashboard* desplegado para monitorear métricas de consumo de recursos de la VIM [57].

En este *dashboard* se aprecian trazas en tiempo real de recursos como memoria, CPU, memoria *swap*, almacenamiento en disco y tráfico de red. Grafana también puede ser usado como intermediario para consultar métricas en Prometheus. Como se discute en el siguiente capítulo, los datos deben ser adquiridos en tiempo real para enriquecer un algoritmo de ML. Disponer de dos métodos de consulta, expande las opciones y hace al escenario de experimentación más flexible.

2.3.3. Gestión mediante Terraform

Terraform es una herramienta para ajuste, gestión de versiones y despliegue de infraestructura. Usando los archivos de configuración en formato `.tf`, es posible ejecutar cualquier tipo

de aplicación sobre la infraestructura de un *datacenter*. Con Terraform, se define un plan que describe cuál es el estado deseado al momento de desplegar una aplicación y, cuando se ejecuta, se encarga de usar la infraestructura subyacente para desplegar la aplicación y lograr el estado deseado. La infraestructura gestionada por Terraform, incluye recursos de bajo nivel como CPU, RAM, disco o recursos de red, así como componentes de alto nivel como DNS (Domain Name Server) o características SaaS (Software as a Service) [39].

Se suele interpretar a Terraform como un mecanismo de IaC (Infrastructure as Code). Cuando se integra con OpenStack, puede usarse para orquestar la infraestructura sin necesidad de interactuar con *Horizon*. Es importante mencionar que, aunque con Terraform se pueden cumplir requerimientos de orquestación, no nace como un orquestador y, por lo tanto, no puede ser comparado con herramientas como OSM (Open-Source MANO) [58] u ONAP (Open Network Automation Platform) [59]. En otras palabras, Terraform cumple con los requisitos de gestión necesarios para ejecutar políticas de autoescalamiento sobre una VIM, con la ventaja de ser liviano y fácil de configurar. La instalación, integración con OpenStack y configuración de Terraform, pueden consultarse en el Anexo D. En este capítulo se crea úncicamente una VM que exporta métricas hacia Prometheus. No obstante, en el Capítulo 4, se detalla cómo puede usarse Terraform en conjunto con un algoritmo de predicción y algunos paquetes para estresar un servidor, para autoescalar recursos de forma proactiva. Como resultado del procedimiento, la interfaz mostrada en la Figura **2-5** evidencia la creación automatizada de una instancia usando Terraform.

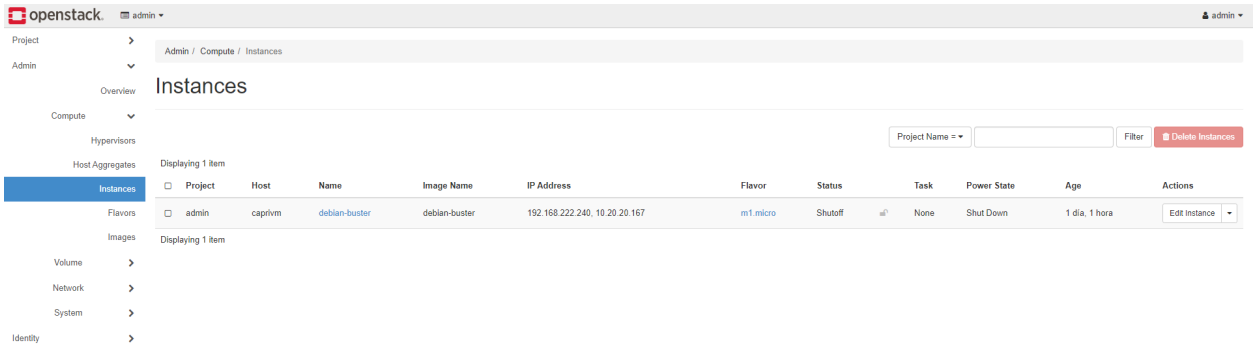


Figura 2-5.: Creación de una instancia en OpenStack desde Terraform.

La instancia puede ser accedida vía SSH usando la IP dinámica asignada por OpenStack (en la Figura **2-5** se observa que es `10.20.20.167`) y la contraseña configurada en el archivo *cloud init* (en este caso `debian`). Es importante mencionar, que al tratarse de una VM, posee un OS completo, por lo cual, cualquier comando Linux/Debian puede ejecutarse.

2.4. Discusión

En este capítulo se propuso un escenario de experimentación para computación en la nube de bajo costo computacional, asequible y local, implementado mediante el uso de tecnologías de código abierto para la gestión de recursos virtuales como OpenStack, Docker y Terraform. OpenStack fue desplegado a través de la distribución MicroStack provista por Canonical, mientras que Docker fue usado para la gestión de los contenedores de Prometheus y Grafana que, en conjunto, componen un sistema liviano de monitoreo y almacenamiento histórico de métricas. Por otro lado, Terraform fue usado como herramienta de gestión de infraestructura, con la ventaja de servir como habilitador para IaC.

A través de la revisión de literatura, se pudo evidenciar que, aunque los autores suelen mencionar cuál es el hardware y software usado para sus investigaciones en mecanismos de autoescalamiento, no ahondan en detalles técnicos sobre cómo se podría desplegar un escenario de experimentación idéntico que permita llegar a resultados similares. Por este motivo, en este capítulo se propone, describe e implementa un escenario de experimentación local y asequible usando tecnologías de código abierto ampliamente desplegadas en entornos productivos de operadores de redes de telecomunicaciones. El escenario está diseñado con base en una arquitectura NFV y sobre éste se pueden aplicar conceptos de bucle cerrado como SON, CLA o *Zero-Touch operation*. Aunque puede ser usado para varios propósitos asociados a temas de operación de infraestructura, en el contexto de esta investigación, es la base para la ejecución de experimentos de autoescalamiento de recursos de manera proactiva. La arquitectura posee un componente que añade la inteligencia predictiva necesaria para el escalamiento de recursos. En detalle, este componente se sustenta en un algoritmo que explota las propiedades estructurales del neocórtex biológico para predecir tendencias como se describe en el siguiente capítulo.

El hecho de que el escenario sea liviano en términos de los recursos de cómputo necesarios para su implementación, puede implicar que aplicaciones de alta demanda no puedan ser ejecutadas sobre éste. Por esta razón, como trabajos futuros, se propone la integración de múltiples VIM en el mismo escenario, análogo a un escalamiento horizontal de los recursos disponibles. La integración puede ser orquestada con Terraform o bien, haciendo uso de otros orquestadores como OSM [58] u ONAP [59]. Sin embargo, debe considerarse que el despliegue de este tipo de orquestadores implica añadir más recursos al escenario de experimentación.

3. Algoritmo de predicción de series de tiempo basado en redes neuronales para la estimación de la demanda de recursos en entornos NFV/SDN

3.1. Introducción

Las técnicas de inteligencia artificial (AI, *Artificial Intelligence*) aplicadas a problemas de redes de telecomunicaciones, han habilitado tres características fundamentales: eficiencia operacional, mejora de la seguridad y escalabilidad de un nuevo catálogo de servicios de ultra-baja latencia para soportar casos de uso como internet de las cosas (IoT, *Internet of Things*), vehículos autónomos, realidad virtual y realidad aumentada. También se habilitan nuevas vías para la solución de problemas comunes en redes como lo son la gestión de fallas, analítica en amenazas y ciberseguridad, mejora en la experiencia del cliente, optimización de tráfico y redes autoorganizadas o SON [60].

En un mundo donde no hay intervención humana para la mayoría de las actividades cotidianas, las tareas y funciones que hacen las personas quedan delegadas a sistemas autónomos o sistemas inteligentes. Este es el caso de las telecomunicaciones, donde los elementos de red prometen ser autónomos en sus decisiones. Bajo este escenario, las experiencias enriquecen un mundo provisto con *machine learning* e inteligencia artificial [61]. ML se refiere a un programa computacional o software que puede aprender de los datos para predecir un estado o condición futura. Por otra parte, los sistemas de AI, más que una forma avanzada de un sistema de ML, están disponibles para actuar y optimizar continuamente sus acciones a lo largo del tiempo y en cualquier contexto. Tradicionalmente, ML es considerado un subcampo de AI. Una analogía para establecer la relación entre ML y AI yace en un juego de pasos. ML es equivalente a predecir el próximo mejor movimiento dado un histórico de movimientos. Esta es una decisión instantánea. AI es equivalente a formular una estrategia, o un proceso, para ganar el juego, lo que involucra la optimización de la secuencia de mejores posibles pasos. Además, esta estrategia puede ser dinámica y actualizarse en cada movimiento [60].

Sin que fuera previsto de ese modo, las tecnologías que promueven la virtualización, principalmente NFV y SDN, se acoplan de forma transparente para conformar lo que se ha denominado redes autónomas [5]. Si bien NFV/SDN consiste en virtualizar la red haciendo que las funciones se ejecuten como software en una infraestructura que hace las veces de nube, las técnicas de AI analizan los datos recolectados para mejorar la automatización de la red mientras se obtiene información útil de la misma. Por esta razón, una red autónoma es un tipo de red resiliente, que hace autodiagnóstico de su estado (*self-healing*), desempeña tareas de autoaprendizaje (*self-learning*) y de autogestión de recursos (*self-optimizing*), análogo a una red SON [62]. Este tipo de redes se basan en la premisa de recolectar y analizar datos para tomar acciones que mejoren el estado actual. Las acciones pueden ir desde optimizar rutas de tráfico e identificar patrones anómalos en la seguridad de la red, hasta predecir fallas de una función de red y balancear el tráfico con base en estas anomalías [20].

Este último punto es de interés en esta investigación. Usando AI, es posible mejorar el S3P (Stability, Security, Scalability and Performance) en una red autónoma [63]. En el caso del desempeño de la red, se trata de identificar patrones anómalos para tomar acciones. Por ejemplo, la memoria y CPU usadas en una máquina virtual (VM, *Virtual Machine*) pueden ser datos tomados para entrenar algoritmos basados en AI con el objetivo de predecir cuál va a ser la salud de esa VM, en un periodo de tiempo determinado. Si se detectara una posible falla, esta puede ser corregida mediante la migración de tráfico mientras se reinicia la VM o se establecen las causas de la anomalía, bien sea de forma automática o manual. Por esta razón, la inteligencia añadida a la red es capaz de tomar información de los recursos (usuarios, dispositivos, redes o aplicaciones) para entregar el mejor servicio en el ambiente en el que opera, ya sea cuando ocurra una degradación (reactivo) o cuando se prediga (proactivo).

Los enfoques predictivos han mostrado tener un alto valor comercial, pues mantienen una operación confiable asegurando el servicio más tiempo que si se usara un enfoque reactivo [29]. Entre la amplia gama de algoritmos de predicción existentes, elegir el más adecuado para la predicción de anomalías, se vuelve una tarea crucial. Los criterios de selección entre algoritmos de predicción pueden variar, sin embargo, es común encontrarse que este tipo de algoritmos deben cumplir con los siguientes puntos [64]:

- Aprendizaje continuo o en tiempo real. Además, el algoritmo debe aprender sin la necesidad de almacenar datos en paralelo.
- Predicciones de alto orden, es decir, que consideren entradas anteriores para realizar la predicción. También, las predicciones deben minimizar los falsos positivos y falsos negativos.
- Efectuar múltiples predicciones considerando probabilidades de ocurrencia. Además,

debe ser capaz de comparar estas predicciones con la entrada actual para detectar anomalías.

- Robustez al ruido de entrada. Además, debe ser capaz de adaptarse a ambientes dinámicos donde la serie de tiempo puede tener comportamientos no estacionarios.

- Tolerancia a fallas como, por ejemplo, ausencia de una entrada.

- Poca o ninguna intervención para ajuste de parámetros del algoritmo, esto es, el algoritmo debe ejecutarse en una faceta automática.

En este capítulo se presentan las investigaciones más relevantes en cuanto al uso de técnicas de predicción para resolver problemas asociados a redes de telecomunicaciones. Durante la revisión, se priorizan las investigaciones que consideran el uso de HTM (Hierarchical Temporal Memory) en sus experimentos. También, se presenta un marco conceptual para entender el funcionamiento interno de HTM. Para finalizar el capítulo, se presenta una implementación de este algoritmo para predecir el comportamiento de una serie de tiempo para pruebas, construida usando información de tráfico de una red móvil real. Este capítulo aborda el primer objetivo específico.

3.2. Trabajos relacionados que consideran técnicas de machine learning para predecir recursos en redes de telecomunicaciones

El uso de técnicas de ML con enfoques predictivos, ha cobrado mayor importancia gracias al enorme crecimiento en la disponibilidad de datos de *streaming* orientados a series de tiempo. Como se mencionó, este tipo de datos posee desafíos técnicos que conllevan a oportunidades en el procesamiento de la información, siendo eje central el aprendizaje no supervisado y en tiempo real [65]. Son muchas las técnicas que han sido aplicadas para procesar información en *streaming* con el objetivo de predecir.

En la literatura se encuentra un gran número de trabajos dedicados a la predicción de demanda de ancho de banda en redes fijas o móviles. Los trabajos en [66, 67] resumen enfoques que se basan en el modelamiento del comportamiento del tráfico. Para esto, se consideran perfiles de tráfico construidos a partir de información de los protocolos de comunicación, donde se explotan características temporales de los paquetes para estimar la capacidad disponible. Aunque los perfiles son construidos usando técnicas estadísticas para el ajuste de curvas, infieren un comportamiento estático en el tráfico que cursa la red [27].

Enfoques más dinámicos han sido propuestos en [68, 69], donde se usan datos históricos de la red para ajustar modelos lineales o no lineales basados en regresiones. La ventaja de estos modelos es que son reajustables según nuevos comportamientos en los datos. Sin embargo, los modelos carecen de un aprendizaje en línea y no son tolerables al ruido en la entrada. En [25] se propone un enfoque predictivo para predecir congestión en redes LTE. Los autores proponen dos modelos, uno persistente y otro autorregresivo derivado del ARIMA (Autoregressive Integrated Moving Average). Los resultados muestran que ambos modelos predicen el comportamiento de la red con alta precisión, pudiendo anticipar hasta 30 horas de comportamiento, suficiente para plantear estrategias SON. A pesar de los buenos resultados, los autores en [26] sugieren que los modelos autorregresivos hacen parte de *frameworks* clásicos de predicción en los que hay dependencia a componentes estacionarios en las series de tiempo.

Para solventar el problema de dependencia, sugieren usar técnicas de ML. Inicialmente, en [70] los autores analizan el uso de ANN (Artificial Neural Networks) en problemas de predicción. Para esto hacen una comparación entre MLP (Multilayer Perceptron), RBF (Radio Base Function), GRNN (Generalized Regression Neural Network) y RNN (Recurrent Neural Network). En el escenario planteado por los autores, el modelo de mejor desempeño es el RBF, seguido de RNN y luego MLP, con GRNN como el modelo con peor desempeño. Luego, los autores en [26] optan por usar modelos de redes neuronales profundas (DNN, *Deep Neural Network*) para predecir nuevamente la congestión en redes LTE. Con los modelos profundos, se incrementa la capacidad de abstracción de características de la serie de tiempo. En este caso se hace uso de LSTM (Long-Short Term Memory). Los resultados muestran que con este enfoque, se predice hasta 30 horas de comportamiento en la red, con un error 7 veces menor al obtenido usando enfoques autorregresivos.

Con el alto desempeño de los modelos LSTM, es usual encontrar su uso para predicciones de series de tiempo, en todo tipo de investigaciones. En [71, 72], los autores hacen uso de LSTM para predecir flujos de tráfico en redes celulares. El primer trabajo asegura que las características no lineales y correlaciones temporales inherentes del tipo de dato procesado, potencian el uso de LSTM para obtener desempeños más altos que otros modelos basados en *autoencoders*. El segundo, usa información del PDCCH (Physical Downlink Control Channel) de la red LTE, para minimizar errores de predicción en comparación con modelos MLP. Sin embargo, ambas investigaciones coinciden en que el uso de LSTM implica mantener una copia de datos almacenada para reentrenamientos de la red. Así mismo, si el enfoque fuera en *streaming* o tiempo real, los datos deberían almacenarse en paralelo y la red debería ser entrenada cada cierto tiempo para ajustarse a los nuevos comportamientos. Por este motivo, LSTM, a pesar de tener un alto desempeño prediciendo, tiene falencias cuando el enfoque de procesamiento de datos es en *streaming*, esto es, datos que son generados continuamente de diversas fuentes de forma simultánea y procesados de forma secuencial o en ventanas de tiempo graduales [73].

Los enfoques en *streaming*, suelen usar LSTM con módulos de reentrenamiento. En estos casos, se suele definir cada cuánto se reentrena la red con un número de muestras. Por ejemplo, LSTM-1000 quiere decir que cada 1000 muestras la red es reentrenada. No obstante, existen enfoques que no requieren reentrenamiento, pues han sido diseñados para procesamiento en *streaming*. Tal es el caso de HTM [65]. HTM es una tecnología de ML inspirada en el comportamiento del neocórtex de los mamíferos. Su estructura de tres componentes, usa codificaciones binarias para el aprendizaje de secuencias reforzadas a través de un aprendizaje Hebbiano [74]. Diferentes autores han hecho una comparación entre algoritmos para aprendizajes de secuencia donde se incluye el HTM [64, 75, 76, 77, 78]. En todas las investigaciones, se ha demostrado que este algoritmo minimiza el error cuadrático medio en varias ventanas temporales de predicción. Sus propiedades van desde el aprendizaje en tiempo real para datos en *streaming*, tolerancia al ruido de entrada, predicción de múltiples puntos posibles, hasta la adaptación a nuevos comportamientos en la serie de tiempo.

El desempeño de HTM ha sido probado en múltiples campos de aplicación. En [79] los autores detectan comportamientos anómalos en tiempo real para vehículos autónomos cuando están en movimiento. Usando el HTM, los autores predicen, clasifican y detectan las anomalías con un 98 % de precisión. Un aspecto importante que introducen los autores, es el monitoreo de múltiples series de tiempo en paralelo. En [80] los autores usan HTM para detectar anomalías en una Smart Grid que genera gran cantidad de datos en tiempo real. Los autores demuestran que HTM detecta tanto anomalías temporales como espaciales, aprendiendo incluso el comportamiento de diferentes días de la semana. Los autores demuestran que HTM tiene mejor desempeño que algoritmos como el *Random Cut Forest*, el *Bayesian Change Point* y el *Relative Entropy*. En [81] los autores usan el HTM como base de un sistema de detección de intrusos (IDS, *Intrusion Detection System*) para ciberseguridad. En este, los autores se enfocan en ataques de múltiples etapas o multistage. Estos ataques son difíciles de detectar porque plantean relaciones lógicas entre etapas, son eventos de baja probabilidad, algunos son fácilmente etiquetables como tráfico normal y hay redundancia en los datos. Por este motivo, los autores introducen un método de preprocesamiento para refinar los datos en *streaming*. Luego, usan HTM para mejorar el aprendizaje de secuencias de los datos refinados y proponen un método de clasificación de anomalías que mejora la precisión en el escenario de interés.

HTM ha sido probado con distintas fuentes de datos, demostrando que es eficiente para detectar anomalías temporales o espaciales, incluso en presencia de ruido. En esta investigación, HTM es usado para predecir valores de CPU, RAM y recursos de red en una VM que soporta una función de red, de manera que sea posible anticipar cuándo se necesita un reajuste en los recursos asignados para que no se afecte la calidad en la entrega del servicio. En la siguiente subsección, se explica con más detalle cuál es el soporte teórico de HTM, sus principales componentes y algunas limitaciones del algoritmo.

3.3. Hierarchical Temporal Memory

HTM es una tecnología de ML basada en la explotación de propiedades algorítmicas y estructurales del neocórtex biológico [82]. El neocórtex es una parte del cerebro de los mamíferos que involucra una gran cantidad de funciones como la percepción sensorial, movimiento consciente, el habla y las habilidades matemáticas. HTM, como el neocórtex, consiste en regiones idénticas conectadas entre sí en una jerarquía. A medida que se sube en la jerarquía, la información de varias regiones converge en una sola y viceversa. La conexión intra regional también es posible. Esta estructura permite representaciones a alto nivel formadas de datos sensoriales de bajo nivel. Así, los objetos heredan propiedades de sus subcomponentes. Esta es la razón por la que el cerebro no memoriza, por ejemplo, todos los posibles vehículos para reconocer cuándo está en frente de un vehículo. En su lugar, memoriza las propiedades que definen un vehículo. La Figura **3-1** representa un acercamiento al neocórtex y los componentes base de cada región.

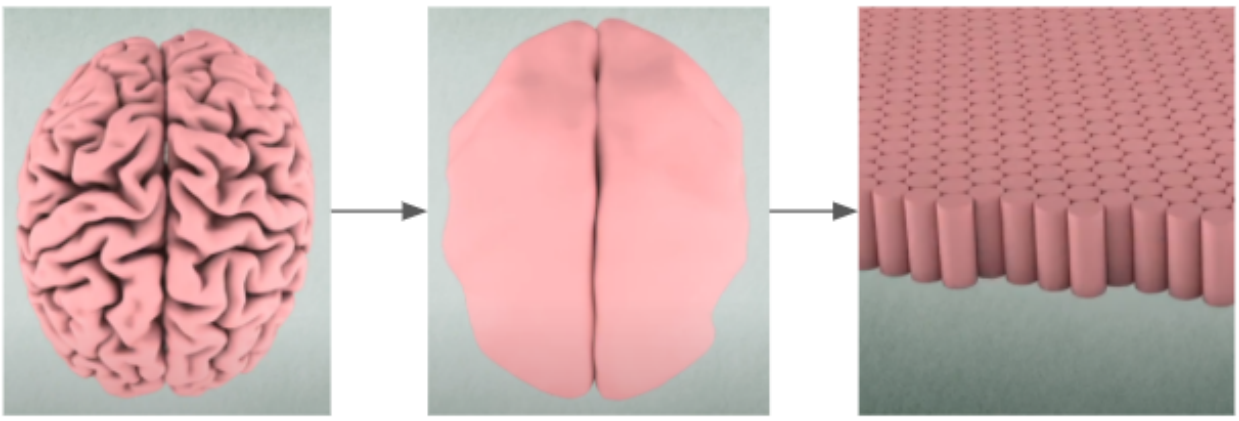


Figura 3-1.: Acercamiento a los componentes del neocórtex [83].

Las regiones del HTM, en cambio, consisten en neuronas interconectadas en columnas. Las conexiones neuronales, tanto entre regiones como internamente, junto a la conexión con los datos de entrada, constituyen lo que se denomina segmentos dendríticos. Hay dos tipos de segmentos en HTM: proximales, que conectan una región con el espacio de entrada o datos de entrada, y distales, que determinan las conexiones entre regiones. Al mismo tiempo, cada segmento dendrítico procesa un conjunto de sinapsis potenciales, cada una de las cuales se conecta a otra neurona o a un bit del espacio de entrada. Los modelos HTM usan sinapsis binarias, es decir, están conectados o desconectados. La condición de sinapsis se determina a través de un valor de permanencia. Si está por encima de cierto umbral, la sinapsis está conectada con un peso igual a la unidad o, en caso contrario, su condición es desconectada con un peso igual a cero. Si el bit del espacio de entrada que corresponde a una sinapsis conectada está activo, entonces la sinapsis se

considera activa o, en caso contrario, inactiva. La Figura **3-2** muestra el modelo de una neurona HTM comparado con una representación biológica. En este punto es importante aclarar, que las conexiones proximales son análogas con las conexiones de *feedforward* en la imagen, mientras que las distales hacen referencia a las conexiones context. El *feedback*, por su parte, es introducido por el algoritmo para hacer correcciones en el aprendizaje del mismo.

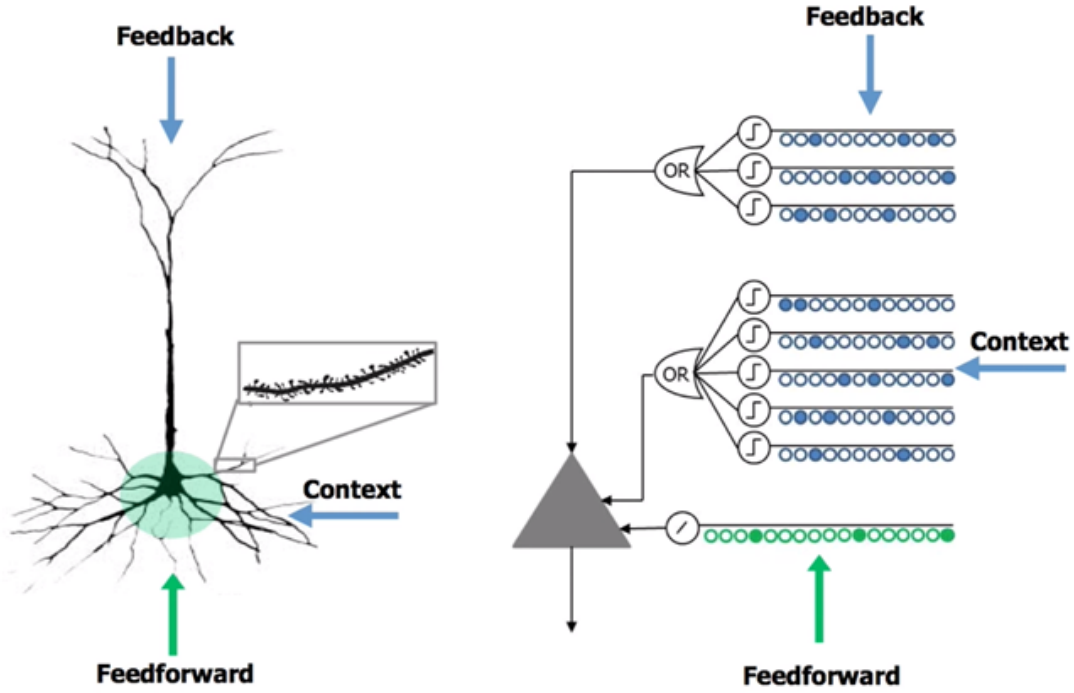


Figura 3-2.: Modelo de neurona HTM comparado con una neurona biológica [64].

Los segmentos dendríticos también pueden determinar el estado de una neurona. Supóngase que una neurona está inactiva. A través de la actividad de los segmentos proximales o debido a conexiones de compensación, la neurona entra en estado activo. Por otro lado, debido a la actividad de los segmentos distales, la neurona puede entrar en un estado predictivo. Es importante mencionar que el HTM usa solo un segmento proximal por columna y la actividad de una columna está definida por el número de sinapsis activas del segmento [77]. Por otra parte, la adición de estados predictivos en las neuronas, habilita el HTM para funcionar como predictor en secuencias o series de tiempo. Para esto, se ha propuesto una arquitectura de predicción de cinco componentes, de los cuales tres componen el núcleo de un sistema HTM. Los componentes son *Encoders*, *Spatial Pooler*, *Sequence Memory*, *Prediction Error* y *Anomaly Likelihood*. Los tres primeros son el núcleo del HTM. La Figura **3-3** muestra una arquitectura con estos componentes dispuestos para tareas de predicción y detección de anomalías, los cuales se describen en los siguientes numerales.

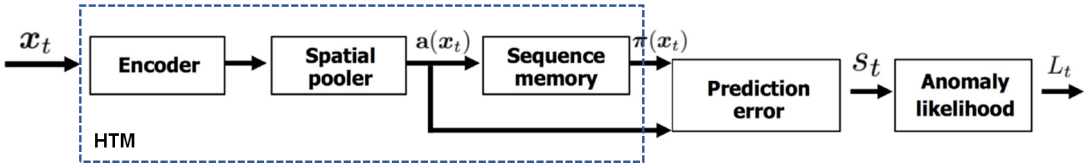


Figura 3-3.: Arquitectura para tareas de predicción usando HTM [65].

3.3.1. Encoders

Los codificadores o *encoders* transforman los datos en vectores de tamaño fijo, para después, convertirlos en una representación binaria denominada SDR (Sparse Distributed Representation). Dado que la mayoría de los datos no son representados en modo binario, los codificadores son necesarios para acoplar el sistema con el HTM. A grandes rasgos, un codificador asigna un SDR diferente a cada punto en los datos, por lo que dos puntos similares son proporcionales según el número de bits activos que se superpongan entre ambas representaciones. Muchos codificadores han sido propuestos en [84], la mayoría para tipos de datos donde el concepto de similitud entre puntos de datos es claramente definido, como números o geolocalizaciones. Por ejemplo, enteros entre 0 y 100 pueden codificarse a través de una ventana de m bits activos en un arreglo de tamaño n. Los enteros adyacentes tendrán una cantidad de bits superpuestos igual a la distancia entre las ventanas de codificación, como se muestra en la Figura **3-4**.

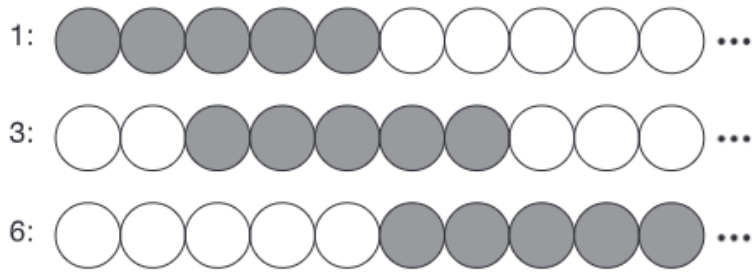


Figura 3-4.: Ejemplo de codificación para números enteros [76].

Otros codificadores más complejos, por ejemplo, para palabras o texto en general, pueden ser consultados en [84]. Otro tipo de datos como audio o video, no tienen propuestas de codificación donde se demuestre su efectividad. La razón es que la transición entre imágenes o notas, contiene información que debe considerarse durante la codificación, complicando la tarea.

3.3.2. Sparse Distributed Representation

La representación distribuida de baja densidad (o escasa), SDR por sus siglas en inglés, es el espacio de representación en el que fluye la información por la arquitectura propuesta en la Figura **3-3**. SDR es un vector de bits del cual solo un pequeño porcentaje (usualmente de 2 % a 4 %) está activo. Este tipo de estructura se asemeja a la representación cerebral de los datos, en la que solo un pequeño porcentaje de neuronas están activas, mientras que el resto permanece inactivo. De allí que la representación se considere distribuida y de baja densidad. La principal característica de SDR, es que puede codificar el significado semántico de la información. Por ejemplo, un conjunto de bits activos puede ser comparado a través de una operación de unión con otro conjunto de bits para inferir si ambas representaciones tienen el mismo significado semántico. A manera ilustrativa, la Figura **3-5** muestra una posible representación SDR. En el ejemplo, muy pocos bits se activan para codificar información que responde a preguntas de sí o no. El objetivo es codificar la información de una bicicleta. Con los bits que están activos, no se puede decir que la codificación no es de un objeto similar a una bicicleta. Sin embargo, podría tratarse también de un triciclo. Así mismo, los bits inactivos descartan la posibilidad de que sea un vehículo con motor. De esta manera, cualquier comparación indagaría en objetos que cumplan esas características. Por otro lado, a medida que se añaden más bits, es posible codificar información más compleja [65].

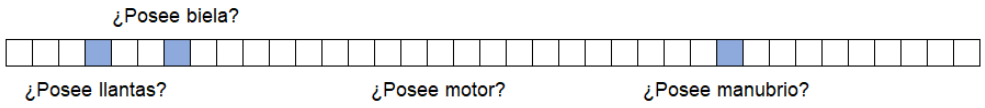


Figura 3-5.: Ejemplo de representación SDR.

Luego de codificar los datos en una representación SDR, se añaden propiedades como:

- Disminución en la probabilidad de desajuste.
- Robustez al ruido de entrada.
- Posibilidad de reducción de costo computacional a través de técnicas de muestro.
- Inclusión de operaciones binarias para la comparación semántica.

SDR mantiene sus propiedades a pesar de que algunos bits se dañen por el ruido. La razón es que existe una probabilidad muy baja para que los bits que fueron dañados se superpongan justo con los del mismo significado semántico esperado. Por otro lado, con SDR se pueden unir decenas de SDR en uno solo, a través de operaciones lógicas como OR, mientras que se pueden comparar con un AND. La cantidad de características o capacidad de representación SDR, vienen dadas por la ecuación 3-1.

$$\text{Capacity} = \frac{n!}{w!(n-w!)} \tag{3-1}$$

donde n es el total de bits usados para SDR y w la cantidad de bits activos. En el ejemplo de la Figura **3-5**, n es igual a 37 y w es 3, por lo que la densidad es del 8.11 %. En una configuración HTM típica, los valores son 2048 para n y densidad del 2 % al 4 %.

3.3.3. Spatial Pooler

Después de codificar los datos en un espacio de representación SDR, la información fluye hacia el *Spatial Pooler* (SP). Este componente garantiza una densidad fija en todos los SDR, la cual suele ser ajustada entre 2 % y 4 % [85]. Para entender la asignación, es importante aclarar que el SP consiste en columnas, cada una conectada a una fracción (típicamente el 50 %) del total de bits del espacio de entrada codificado. Como se mencionó anteriormente, estas conexiones se denominan sinapsis proximales. Cada una de estas sinapsis tiene asignado un valor de permanencia entre 0 y 1, y solo cuando esta sinapsis supera un umbral (por defecto 0.5) se dice que está activa, es decir, puede transportar una señal desde la entrada. La Figura **3-6** muestra las conexiones entre el SP y la entrada. Es importante aclarar que, aunque la representación es bidimensional, el SP consiste en columnas con neuronas que le dan una profundidad que no se aprecia en esta figura.

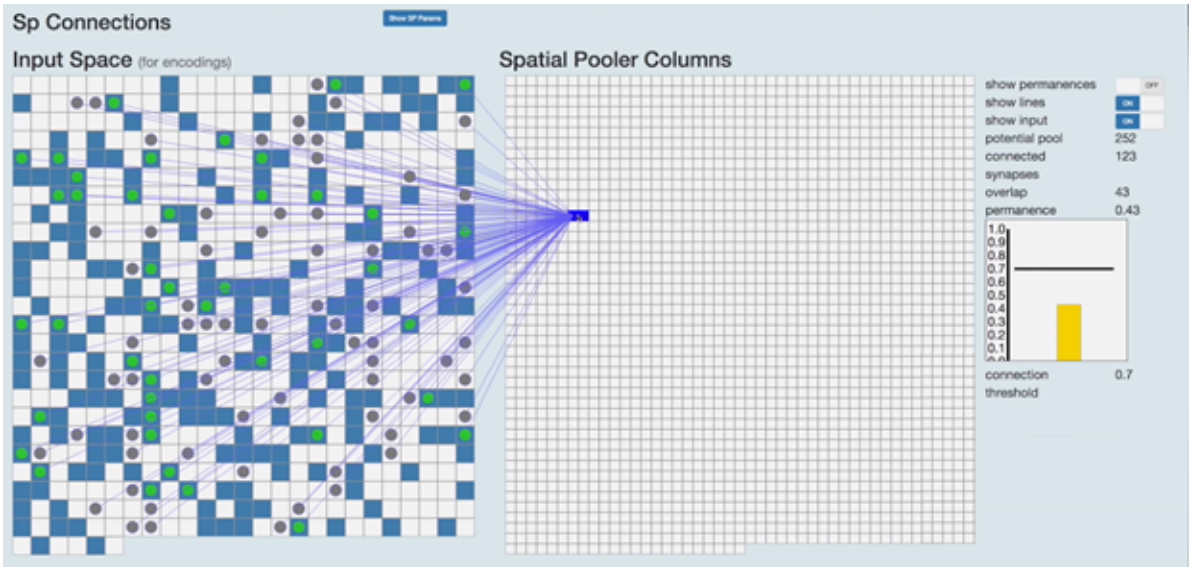


Figura 3-6.: Conexión entre el SP y el espacio de entrada [83].

Dada una entrada, a cada columna se le asigna un puntaje igual al número de bits activos en la entrada, conectados a través de sinapsis activas. El top x de columnas

ordenadas de forma descendente según el puntaje mencionado, son activadas. Con este mecanismo se logra una densidad fija, mientras que las columnas activadas son sintonizadas para reconocer entradas similares a través de un aprendizaje Hebbiano [74], es decir, las permanencias de las sinapsis para bits activos incrementan y las otras disminuyen. El procedimiento se repite para cada nueva entrada. La Figura **3-7** resume el comportamiento del SP.

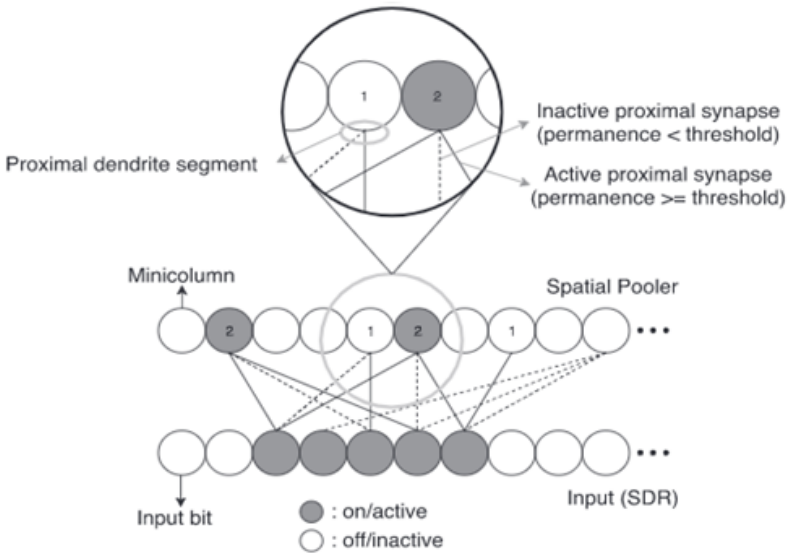


Figura 3-7.: Comportamiento del SP para el aprendizaje [76].

En general, SP convierte las entradas SDR en una salida de densidad fija, también representada en SDR, mientras se adapta al cambio de los patrones en la entrada, sin perder información semántica que relaciona varios puntos de los datos [76].

3.3.4. Sequence Memory

El SDR generado por el SP no tiene información temporal, es decir, una entrada no depende de una secuencia de entradas anteriores. Por esta razón, uno de los componentes clave es *Sequence Memory*, también llamado como *Temporal Memory* (TM). Este componente actúa como una extensión del SP, subdividiendo cada una de sus columnas en un número fijo de neuronas. De este modo, una columna activa equivale a que una o más de sus neuronas están activas. Idéntico a las sinapsis proximales, las sinapsis distales aumentan su peso entre neuronas y están agrupadas en segmentos dendríticos distales. Sin embargo, a diferencia de los segmentos proximales, donde existe exactamente uno por columna, en las conexiones de contexto pueden existir múltiples segmentos distales por columna dependiendo de la cantidad de neuronas. Cuando una columna se activa con una entrada, se propaga una señal a través de las sinapsis distales en todos sus segmentos. Si el número de señales que recibe una neurona en un segmento excede algún umbral configurable, entra en **estado predictivo**. Esto indica

que la neurona espera activarse en la próxima entrada. Así, si la columna se activa, solo sus neuronas en estado predictivo se activan. Si no hay ninguna, todas las neuronas de la columna se activan, lo que en HTM se denomina *bursting*. Cuando una neurona en estado predictivo recibe una nueva entrada, la sinapsis en sus segmentos es reforzada a través de aprendizaje Hebbiano. Para cualquier segmento con pocas sinapsis, las nuevas sinapsis crecen hasta un subconjunto de neuronas elegido al azar, las cuales estaban activas en el paso anterior [76, 82]. La Figura **3-8** muestra una neurona (en azul) y sus conexiones distales (en púrpura), con otras neuronas (en amarrillo) que le dan contexto sobre una nueva entrada. En este caso, la figura ilustra un paso en el aprendizaje de una secuencia de notas musicales [83].

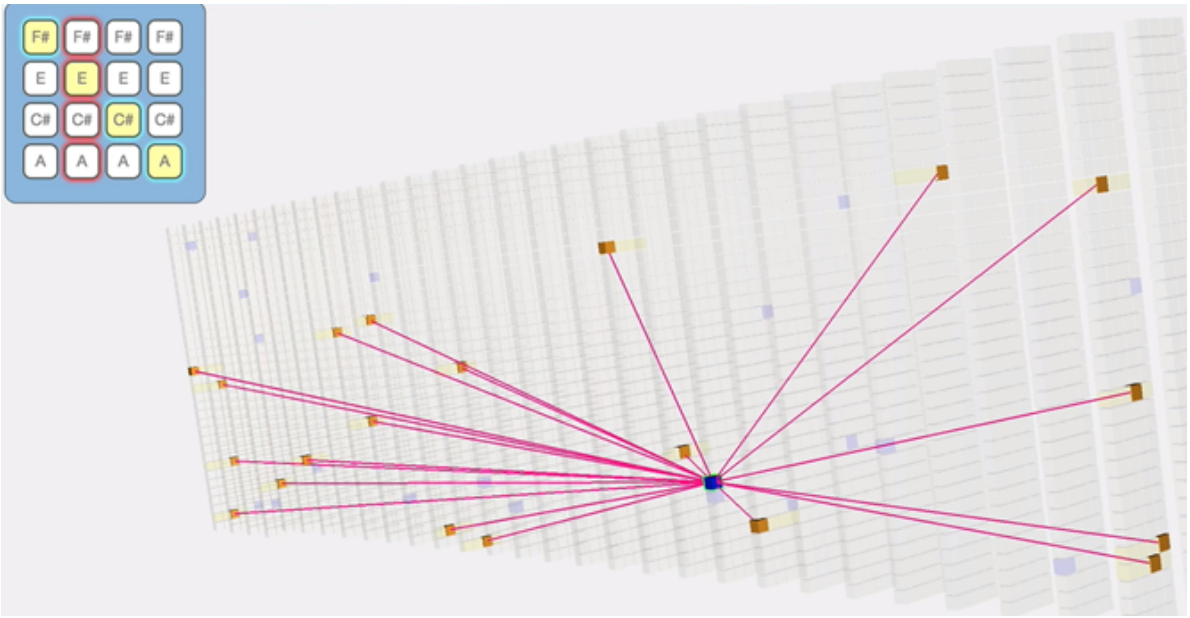


Figura 3-8.: Conexiones distales para la obtención de información de contexto e información temporal [83].

3.3.5. Prediction Error

Para calcular el error de predicción, también llamado *Anomaly Score* en HTM, hay que considerar la notación propuesta en la Figura **3-11**. La entrada actual se representa por x_t, mientras que las entradas anteriores pueden ser $\dots x_{t-3}, x_{t-2}, x_{t-1}$. Todas son codificadas en SDR. Así pues, dada una entrada actual x_t, $a(x_t)$ es una codificación SDR de la entrada actual y $\pi(x_{t-1})$ es un vector SDR que representa la predicción interna realizada por el HTM de $a(x_t)$. La dimensionalidad de ambos vectores es la misma y por defecto es 2048. Ahora, sea el error de predicción, s_t, un valor escalar inversamente proporcional al número de bits en común entre los vectores binarios actual y predicho, está dado por la ecuación 3-2.

$$s_t = 1 - \frac{\vec{\pi}(x_{t-1}) \cdot \vec{a}(x_t)}{\|\vec{a}(x_t)\|} \tag{3-2}$$

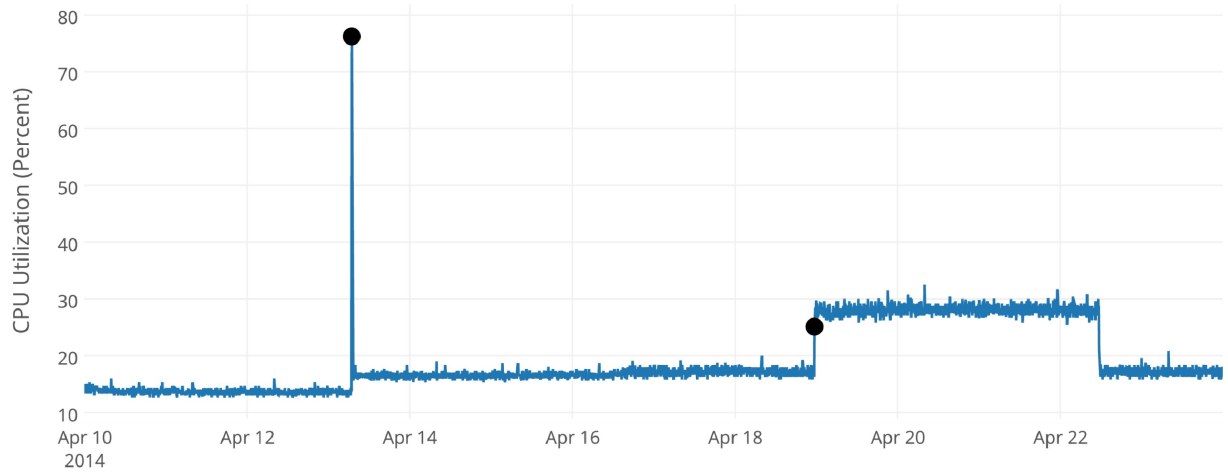
donde $\|\vec{a}(x_t)\|$ es el total de bits en 1 en $\vec{a}(x_t)$. En la ecuación 3-2, el error será 0 si la entrada $\vec{a}(x_t)$ coincide exactamente con la predicción y 1 si ambos vectores son ortogonales, es decir, no tienen bits en común. El error de predicción da una medida de cómo el modelo HTM predice la entrada actual x_t.

Si hay cambios en el comportamiento de la serie de tiempo, el aprendizaje continuo de HTM se adapta, llevando automáticamente a 0 el error de predicción, una vez se conoce el nuevo comportamiento normal. La Figura **3-9** muestra un ejemplo tomado de [65] para el comportamiento de s_t. En el ejemplo, el porcentaje de uso de CPU en un servidor de base de datos, tiene dos anomalías señaladas con puntos negros. Ambas anomalías son temporales, es decir, luego de que ocurren hay un cambio en el comportamiento normal de la serie. En ambos puntos, se observa que el modelo HTM notifica un error de predicción positivo, que luego converge a 0 gracias al aprendizaje en línea del algoritmo. Lo anterior evidencia su capacidad de adaptación a nuevos comportamientos. Por otro lado, es importante mencionar que en HTM hay múltiples predicciones para $\vec{\pi}(x_t)$, las cuales son representadas en la unión binaria de predicciones individuales de un gran número de vectores de baja densidad y alta dimensionalidad codificados en SDR. De este modo, si dos entradas completamente diferentes son posibles y predecibles, sea cual sea la entrada, el error de predicción será cercano a 0. Cualquier otra entrada que se salga del espacio posible o predecible, generará un error positivo, como en el caso del ejemplo de la Figura **3-9**.

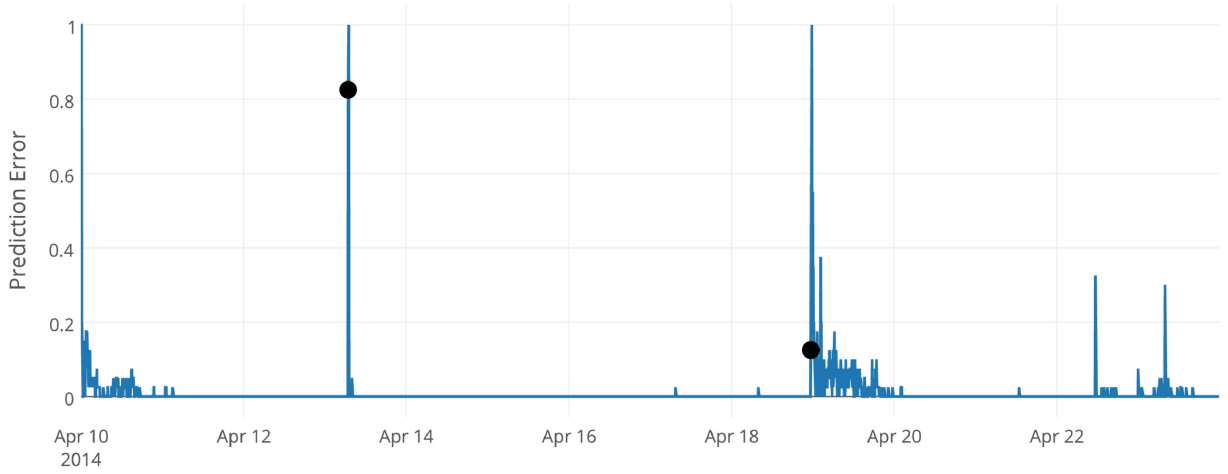
3.3.6. Anomaly Likelihood

El error de predicción resulta ser una medida que funciona bien para escenarios donde la entrada no es ruidosa. Sin embargo, algunas aplicaciones pueden tener comportamientos ruidosos e impredecibles. En el ejemplo de la Figura **3-10** [65], se muestra el comportamiento de la latencia en un balanceador de carga cuando recibe solicitudes HTTP. Aunque la latencia se mantiene en un intervalo relativamente bajo, puede presentar comportamientos aleatorios que llevan a picos en el error de predicción. Tratar de establecer un umbral de anomalías para ese comportamiento del error, puede inferir una alta tasa de falsos positivos. Para manejar este tipo de escenarios, se introduce el concepto de *Anomaly Likelihood* o la probabilidad de anomalía.

La probabilidad de anomalía se considera como una segunda etapa del error de predicción en la que se modela la distribución de valores de error como una métrica indirecta. Esta distribución se usa para observar la probabilidad de que el estado actual sea anómalo. Así



(a) Uso de la CPU en un servidor de base de datos en el tiempo

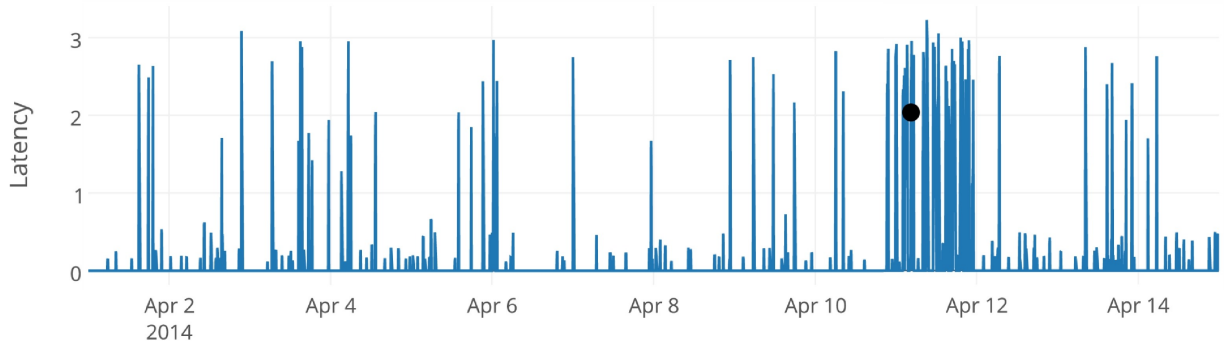


(b) Error de predicción en el tiempo mientras se entrena el HTM sobre la serie

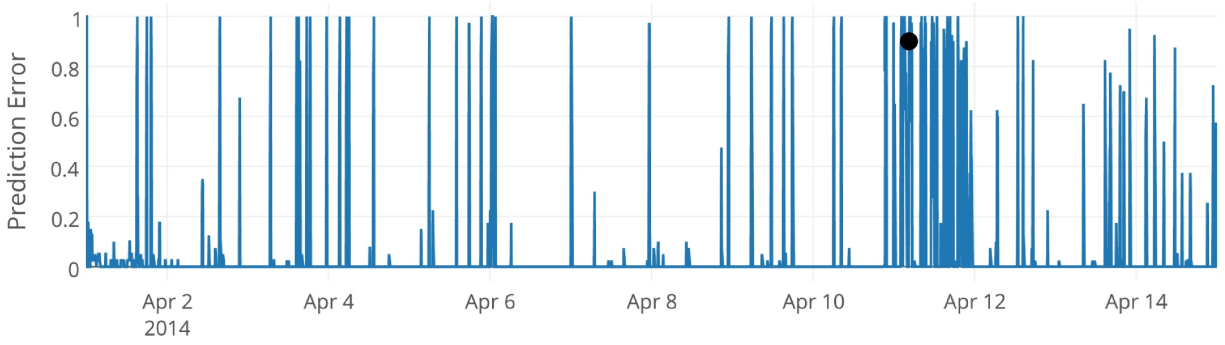
Figura 3-9.: Serie de tiempo ejemplo para la predicción del error [65].

pues, la probabilidad de anomalía es una métrica probabilística que define cuán anómalo es el estado actual basado en el histórico de predicciones del modelo HTM. Para calcular la probabilidad de anomalía, se mantiene una ventana de W valores de error. De este modo, se modela como una distribución normal donde la media μ_t y la varianza σ_t^2 son actualizadas continuamente considerando valores de error previos. Las ecuaciones 3-3 y 3-4 definen la media y la varianza, respectivamente.

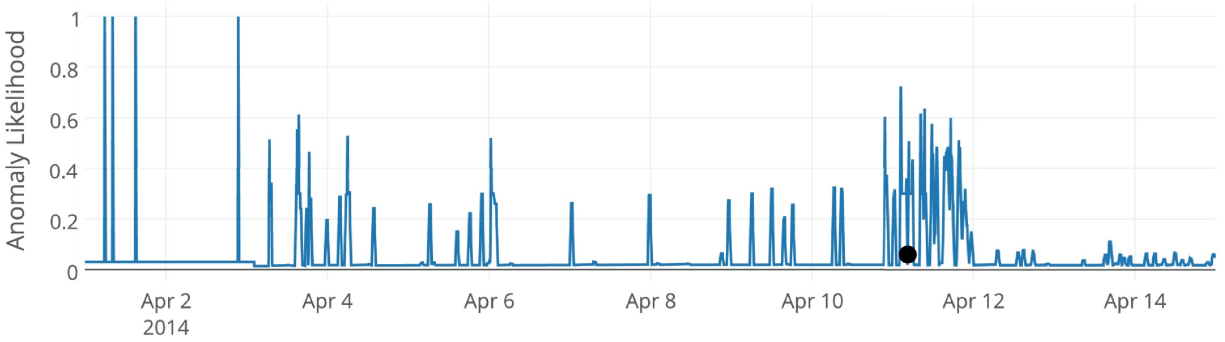
$$\mu_t = \frac{\sum_{i=0}^{i=W-1} s_{t-i}}{W} \tag{3-3}$$



(a) Datos de mediciones de latencia en un balanceador de carga de un servidor web HTTP.



(b) Error de predicción calculado por el HTM para los valores de latencia.



(c) Probabilidad de Anomalía calculada del error de predicción.

Figura 3-10.: Serie de tiempo con mediciones ruidosas [65].

$$\sigma_t^2 = \frac{\sum_{i=0}^{i=W-1}(s_{t-i} - \mu_t)^2}{W-1} \tag{3-4}$$

Luego, se calcula un promedio de errores cercanos al instante actual y se aplica un umbral a la probabilidad de cola Gaussiana (Q-Function [86]) para decidir si se declara o no como

una anomalía. La probabilidad de cola es la probabilidad de que una variable sea mayor que x desviaciones estándar por encima de la media. Así, la probabilidad de anomalía está dada por la ecuación 3-5, mientras que el promedio de errores cercanos está dado por la ecuación 3-6.

$$L_t = 1 - Q\left(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}\right) \tag{3-5}$$

donde:

$$\tilde{\mu}_t = \frac{\sum_{i=0}^{i=W'-1} s_{t-i}}{W'} \tag{3-6}$$

W' es una ventana para una media móvil de observaciones de error cercanas al estado actual. Por regla general, $W' \ll W$. En la práctica se suele configurar $W = 8000$ y $W' = 10$. Para determinar si se trata de una anomalía, se usa un umbral configurable ϵ, por lo que la detección de anomalías está dada por la ecuación 3-7.

$$\text{anomaly detected}_t = L_t \geq 1 - \epsilon \tag{3-7}$$

Establecer un umbral para L_t, donde se involucra una probabilidad de cola Gaussiana, trae un límite superior para el número de alertas y el número de falsos positivos. Si se establece ϵ muy cercano a 0, sería poco probable obtener alertas con probabilidades mucho mayores que ϵ. En su lugar, se ha demostrado que valores como $\epsilon = 10^{-5}$, funcionan bien para una amplia gama de aplicaciones [65]. En la Figura 3.10(c) se demuestra que la probabilidad de anomalía posee picos donde en realidad puede haber una anomalía, diferente al error de predicción cuyas mediciones siguen el ruido de los datos de entrada.

En general, L_t es una medida de qué tan bien el modelo predice los siguientes puntos de la serie de tiempo, relativo a su historia reciente. Cuando los datos de entrada poseen muy poca varianza, los valores de s_t y L_t son idénticos, y cualquier pico en el primero se refleja en el segundo. No obstante, en escenarios con aleatoriedad o ruido, un pico en s_t no necesariamente significa un aumento en L_t. Es importante mencionar, que un escenario donde una serie de tiempo pasa de un estado ruidoso a uno más estacionario, también dispara una anomalía.

3.3.7. Limitaciones de HTM

Los autores en [75] recopilan una serie de limitaciones de HTM. Primero, al tratarse de un algoritmo con acceso solo a la entrada actual, le puede tomar mucho tiempo aprender

secuencias con dependencias a largo plazo, en comparación con algoritmos que tienen acceso a un búfer histórico de información. En teoría, este aprendizaje de secuencias se acelera si se mantiene un histórico de datos y se entrena previamente HTM, pero trae consigo un aumento en los recursos mínimos necesarios para su ejecución, además de contradecir la premisa de aprendizaje en *streaming* del algoritmo. Segundo, aunque HTM es robusto al ruido gracias a su representación distribuida y binaria, el modelo de memoria secuencial es sensible al ruido temporal. En otras palabras, se pierde información de contexto en la secuencia si algunos de sus elementos, durante el entrenamiento, representan ruido. Es importante aclarar, que esto es diferente a añadir ruido entre secuencias pues, en este caso, es como si el algoritmo aprendiera el ruido inducido. Tercero, HTM no desempeña tan bien tareas de aprendizaje gramatical. De hecho, los modelos basados en LSTM han mostrado ser más efectivos en estas tareas. Por último, no se han evidenciado experimentos en los que HTM sea comparado con otros algoritmos para evaluar su desempeño manejando datos multidimensionales como voz o video. En este punto, sería interesante combinar el espacio de representación usado por HTM, con otras técnicas de ML para resolver problemas de aprendizaje de secuencias en datos multimedia.

3.4. Implementación de HTM para predecir volumen y detectar anomalías en el comportamiento del tráfico de una red móvil real

HTM es un algoritmo que ha sido impulsado por Numenta Inc. Como compañía, Numenta realiza investigaciones en neurociencia, potenciando el desarrollo de *frameworks* que imitan comportamientos del cerebro en pro de contribuir al estado actual de AI y ML. HTM es el resultado de evidencias neurocientíficas. Dicho de otro modo, se trata de un algoritmo inspirado y limitado por la biología [87]. Inicialmente, desde Numenta se propuso un *framework* de inteligencia donde se desarrollan las librerías para implementar HTM. Luego, el código fue liberado y se creó toda una comunidad en torno a HTM, que ha venido aportando a la evolución y adopción del algoritmo. A la fecha, hay múltiples formas de implementar HTM; la tomada en esta investigación usa las librerías de Python 3 como sugiere la sección *Building from Source* en [88]. La estructura del algoritmo sigue la arquitectura de la Figura **3-3**. Inicialmente, se definen todos los parámetros de cada elemento de HTM. Luego, se codifica la información secuencial para pasarla por SP y TM. Por último, se estiman los errores de predicción y probabilidad de que el estado actual sea una anomalía. En la Figura **3-11** se propone un diagrama de flujo con las etapas de implementación del algoritmo. En un enfoque en *streaming*, una vez se inician los componentes que implementan HTM, se itera continuamente sobre cada nuevo dato en la entrada, sin necesidad de iniciar nuevamente cada componente.

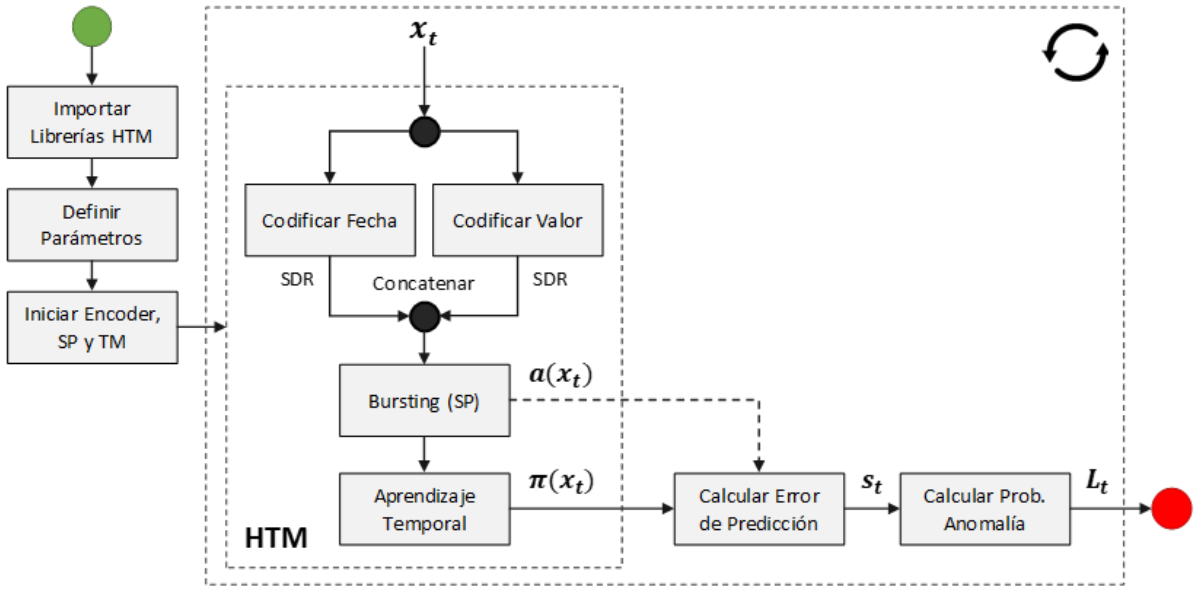


Figura 3-11.: Diagrama de flujo con las etapas de implementación de HTM.

Para demostrar las capacidades de HTM, se usa una secuencia correspondiente a un histórico de un mes del volumen de tráfico de una red móvil real, como la que se observa en la Figura **3-12**. Los datos fueron colectados mediante un NMS (Netowrk Management System) que envía periódicamente consultas SNMP (Simple Network Management Protocol) a la interfaz de red del CSR (Cell-site Router) ubicado en un eNB de la red LTE. Luego, los datos son ingestados y procesados en una arquitectura de *Big Data* idéntica a la propuesta en [89].

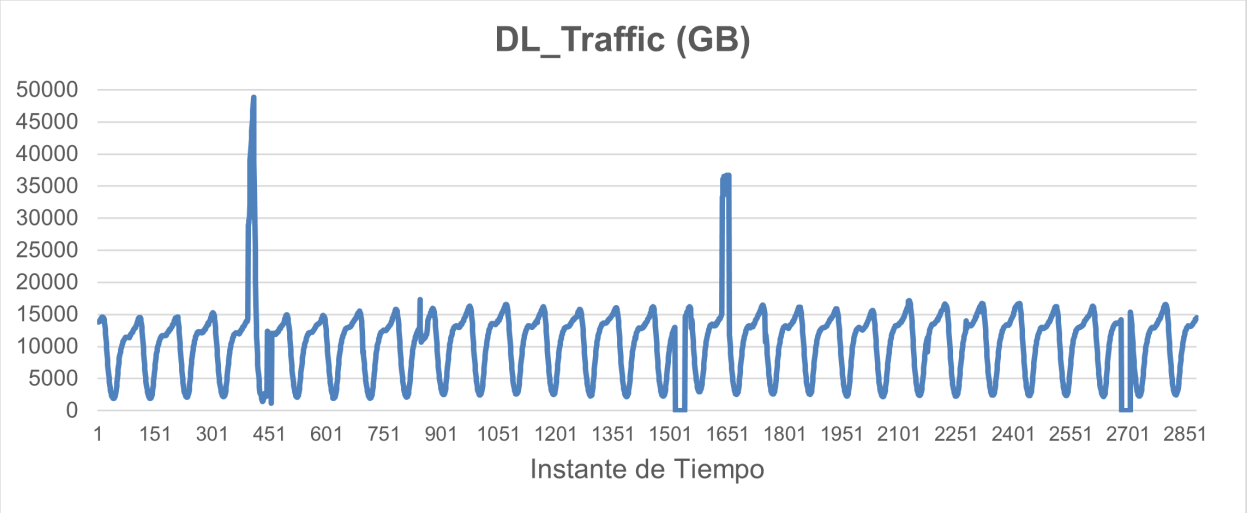
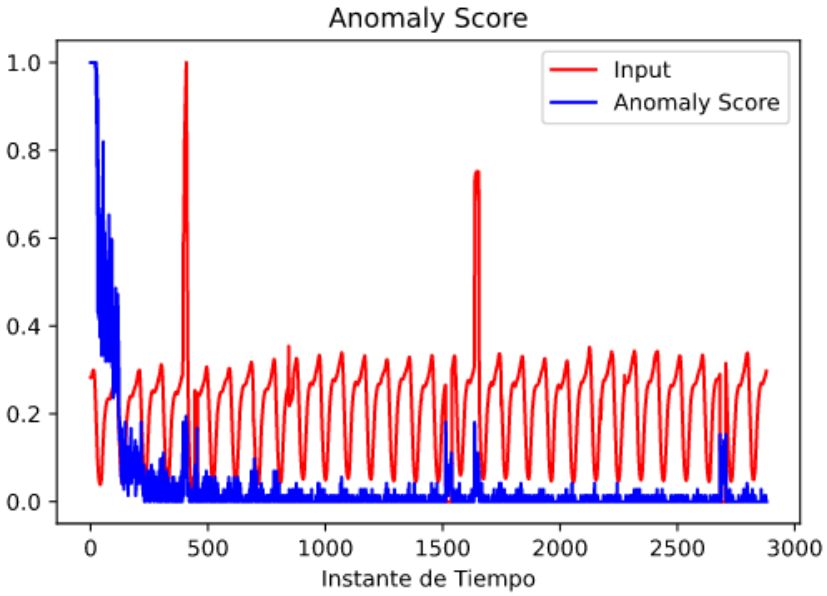


Figura 3-12.: Serie de tiempo usada para la experimentación.

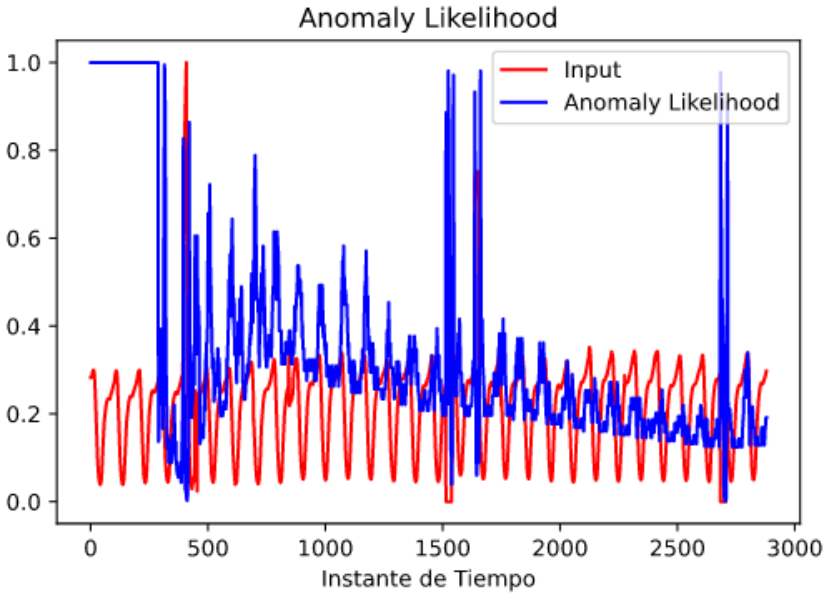
La granularidad de los datos es de 15 minutos, por lo que hay cerca de 2880 muestras que representan 1 mes de información. Intencionalmente, se añaden dos picos y dos valles como comportamientos anómalos para evaluar el desempeño de HTM durante su detección. Además, se proponen dos escenarios de predicción: uno donde se predice 15 minutos y otro donde se predice 1 hora. Para simular el comportamiento en *streaming*, cada punto de la serie de tiempo es recorrido individualmente manteniendo el aprendizaje activo. De esta forma, el SP varía las columnas activas mientras TM aprende la secuencia. Cuando el algoritmo reconoce el comportamiento normal, la detección de las anomalías inducidas es más eficiente. En la Figura 3.13(a) se muestra el comportamiento del error de predicción, mientras que en la Figura 3.13(b) se muestra el comportamiento para la probabilidad de anomalía.

Ambas gráficas tienen un comportamiento en el que, al inicio, los valores que indican una anomalía son relativamente altos debido a que aún no se ha aprendido el comportamiento normal de la secuencia. Mientras esto sucede, se evidencia cómo los valores convergen a valores más pequeños. Por ejemplo, en la Figura 3.13(b) se observa que, en la anomalía cercana a la muestra 1600, los valores de la probabilidad de anomalía superan el umbral de 95 %, mientras que en las últimas muestras de la secuencia, el valor se aproxima a 20 % dado que el comportamiento normal ha sido aprendido. También, se puede ver que en todas las anomalías inducidas, la probabilidad de anomalía supera el umbral de 95 % evidenciando un comportamiento atípico. Por otro lado, si la detección de anomalías se desea hacer automáticamente, es necesario un umbral configurado por el usuario [65]. En la práctica, se establecen umbrales de la probabilidad de anomalía por encima del 95 % de probabilidad.

Al momento de predecir, la Figura 3.14(a) muestra el resultado para 15 minutos, es decir, un instante de tiempo hacia adelante, mientras que la Figura 3.14(b) lo hace para 1 hora. En el primer caso, se observa cómo el algoritmo aprende rápidamente nuevos comportamientos, prediciendo acertadamente la tendencia en la primera anomalía inducida. Sin embargo, al siguiente día, considera que esa anomalía es parte del comportamiento normal y la predice nuevamente. El mismo comportamiento sucede para las siguientes anomalías. Por ejemplo, cerca del instante de tiempo 1500, se evidencia que el algoritmo se ajusta rápidamente al comportamiento anómalo, prediciendo el valle un instante después de que ocurra. Sin embargo, al siguiente día vuelve a predecirlo sin que suceda. Esto sugiere, que la predicción se ajusta rápidamente al comportamiento anómalo de la serie de tiempo, pero memoriza comportamientos del día anterior que refleja en sus predicciones, aun cuando no suceden. La misma rapidez de HTM para aprender el comportamiento de la serie se refleja al siguiente día, cuando el pico anterior no es predicho y la predicción se ajusta al comportamiento esperado de la secuencia. En el segundo caso de la predicción de 1 hora, se observa que el algoritmo no se ajusta rápidamente a los comportamientos anómalos y en su lugar considera que van a suceder al día siguiente. Además, esta predicción añade un pico entre los instantes de tiempo 2000 y 2500, que no sucede en la serie de tiempo original.



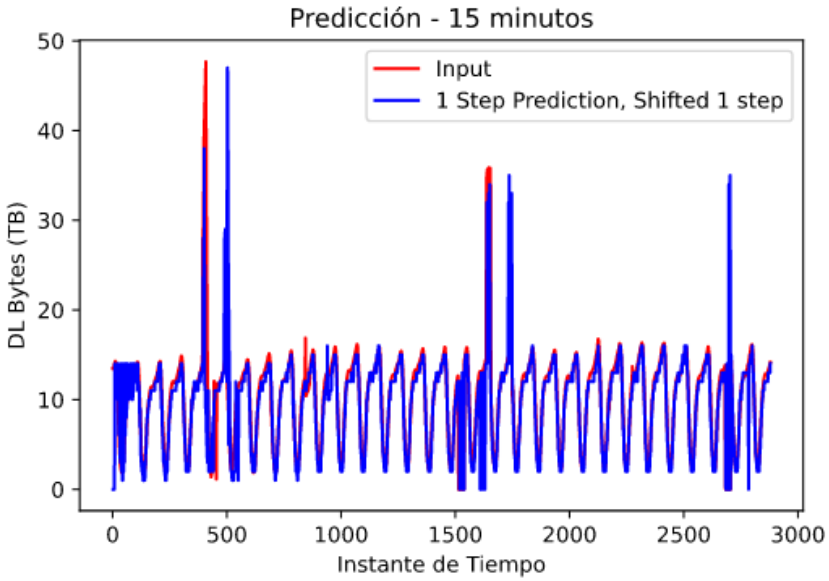
(a) Error de predicción



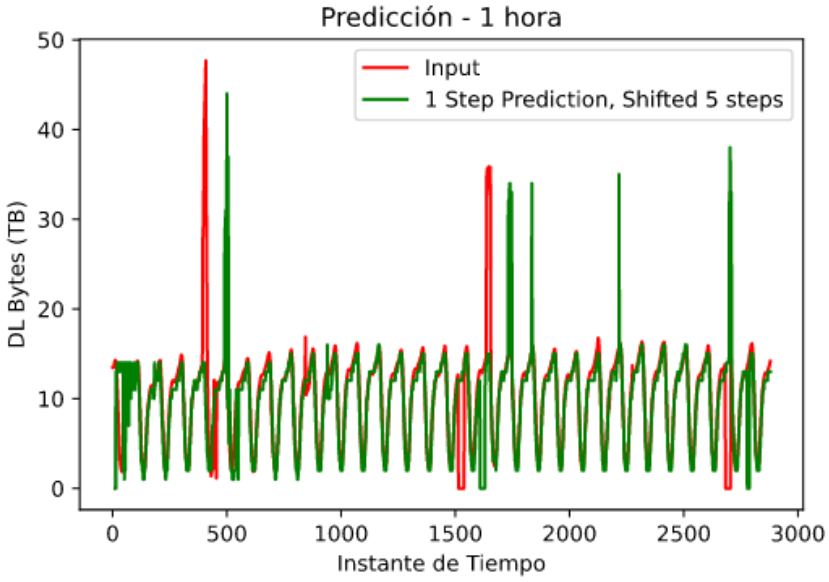
(b) Probabilidad de anomalía

Figura 3-13.: Detección de anomalías usando HTM en la serie de tiempo experimental propuesta.

A pesar de los inconvenientes de las predicciones cuando se presentan comportamientos anómalos en la serie de tiempo, el algoritmo tiene un buen desempeño prediciendo valores en días donde se indujeron anomalías. En ambos casos, se visualiza cómo las predicciones se



(a) Predicción de un solo paso, 15 minutos



(b) Predicción de un solo paso, 1 hora

Figura 3-14.: Detección de anomalías usando HTM en la serie de tiempo experimental propuesta.

superponen con los datos de entrada, permitiendo inferir con eficiencia el comportamiento de la serie de tiempo. Para hacer una comparación cuantitativa de los resultados, se hace uso del error cuadrático medio (RMSE, *Root Mean Square Error*) que puede calcularse mediante la ecuación 3-8.

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_l - y_i)^2}{n}} \tag{3-8}$$

donde $\hat{y}_i$ son los valores predichos, y_i los valores observados y n el número de observaciones. Para la predicción de 15 minutos, el RMSE es de 3,88, mientras que para la de 1 hora, es de 4,47. En [75] usando una serie de idéntica forma, los autores compararon varios modelos obteniendo un MAPE (Mean Absolute Percent Error) de 15,7 % para ELM (Extreme Learning Machine), 9,6 % para ESN (Echo State Networks), 9,9 %, 8,8 % y 8,2 % para LSTM con reentrenamiento en 1000, 3000 y 6000 respectivamente, y 7,8 % para HTM.

3.5. Discusión

En este capítulo se discutió la importancia de las técnicas de ML para predecir comportamientos en las redes de telecomunicaciones cuando se busca autonomía en su operación. Se presentaron trabajos relacionados que consideran diferentes modelos y técnicas usadas para tareas de predicción. Desde perfiles de tráfico hasta técnicas de ML como LSTM, diversos autores han propuesto *frameworks* o metodologías para predecir principalmente la demanda de tráfico en redes. La intención es poder anticiparse a fallas en la red para tomar acciones correctivas antes de la presencia de anomalías. Algunos autores incluso enmarcan el uso de estas técnicas bajo estrategias SON.

En el contexto de esta investigación, se propone el uso de una técnica de ML novedosa, inspirada en el comportamiento del neocórtex de los mamíferos. Esta técnica, impulsada por Numenta, se conoce como HTM y ha demostrado ser eficiente en diferentes ámbitos de aplicación. Para ilustrar la eficiencia del algoritmo, se usa una secuencia que representa la demanda de tráfico de una red móvil real. A la secuencia se añaden dos picos y dos valles intencionalmente para analizar el desempeño de HTM ante comportamientos anómalos. Los resultados muestran, que cuando se predicen ventanas cortas de 15 minutos, el algoritmo se ajusta rápidamente al comportamiento anómalo, pero guarda este estado como si fuera parte del comportamiento normal, prediciéndolo al día siguiente incluso cuando no sucede. En el caso donde se predice una ventana de 1 hora, el algoritmo no se ajusta al comportamiento anómalo. Sin embargo, igual que en el caso anterior, almacena este estado para predecirlo posteriormente, aunque no suceda. Incluso, se pudo evidenciar que la misma predicción introduce un pico anómalo que no tiene un antecedente cercano en la serie de tiempo. A pesar de estos inconvenientes, HTM predice eficientemente el comportamiento normal de la serie de tiempo, con un RMSE de 3,88 en el caso de 15 minutos y 4,47 en el de 1 hora. En el instante en que el algoritmo predice una anomalía, hay un tiempo de guarda para tomar acciones correctivas antes de que ocurra. Esta capacidad de HTM puede ser usada,

en el contexto de redes autónomas, como disparador de una tarea de autoescalamiento, *throttling* o priorización. Con la selección e implementación de HTM se da por cumplido el primer objetivo específico de la investigación. Por otra parte, en el siguiente capítulo se demuestra que predecir, en lugar de esperar una anomalía, reduce los tiempos de respuesta en la infraestructura para tareas de escalamiento de recursos.

4. Implementación de un mecanismo de autoescalamiento de recursos sobre un entorno NFV/SDN basado en OpenStack

4.1. Introducción

El advenimiento de 5G obliga a que las redes móviles sean concebidas de una forma flexible, gestionable, altamente disponible y adaptable al comportamiento dinámico de la demanda de servicios. La transición a esta generación, posibilita la creación de aplicaciones que establecen un nuevo límite de desempeño en redes, como latencias más bajas, mayor confiabilidad en la comunicación, eficiencia energética y mejora de *throughput* [90]. En una arquitectura tradicional, las redes no tienen capacidades de escalamiento automatizadas, pues existe una dependencia hacia el hardware dedicado y una limitación por el acople entre el plano de control y el plano de datos, requiriendo la intervención humana. Mediante las tecnologías NFV y SDN, es posible desacoplar estos planos, además de virtualizar los recursos de la infraestructura, típicamente clasificados en capacidad de procesamiento, almacenamiento y red [91, 92]. Además, estas tecnologías han demostrado ser habilitadoras para aprovisionar servicios de forma dinámica, autoescalable y automática [93].

Tradicionalmente, iterar manualmente sobre posibles formas de asignación de recursos resulta en la asignación óptima de estos. Sin embargo, esto puede incurrir en el incremento de los costos operacionales [94]. Para evitarlo, es necesario automatizar el proceso de asignación, sin sacrificar la alta disponibilidad y la flexibilidad otorgada por la adopción de infraestructuras virtuales. Para esto, los mecanismos de autoescalamiento resultan adecuados para mantener, en valores aceptables, las métricas de desempeño de una infraestructura mientras se distribuye la carga sobre los recursos existentes o los recién creados [95]. Precisamente, en el contexto de NFV, el autoescalamiento es usado para crear o aprovisionar dinámicamente los recursos con base en el requerimiento de desempeño de los servicios y umbrales de disparo. Esto es, la adición o eliminación de recursos se efectúa cuando se supera un umbral de alguna métrica de interés. Como ejemplo, este tipo de mecanismo de escalamiento es usado en herramientas de gestión y orquestación como OSM [96].

Así pues, determinar el umbral de disparo adecuado puede verse condicionado por factores como la cantidad de recursos disponibles o el comportamiento normal de la infraestructura [97]. No obstante, es un método que ha mostrado ser eficaz en la implementación de los mecanismos de autoescalamiento de nubes públicas como GCP y su módulo Autoscaler, AWS y Autoscaling o Azure y Azure Autoscalable. Particularmente, estas nubes ofrecen la opción de escalar con base en un umbral definido por el usuario o automatizar la selección del umbral mediante algoritmos de ML. En el primer caso, si el umbral fue escogido inadecuadamente, el escalamiento puede verse forzado por el comportamiento fluctuante en la demanda de recursos, mientras que en el segundo hay una dependencia hacia la tendencia en la demanda. Si, por ejemplo, la demanda crece diariamente de forma constante pero el umbral es escogido con base en el comportamiento de la última semana, el comportamiento dinámico añadido por las horas pico y valle, puede llevar a la infraestructura a momentos de subdimensionamiento, afectando la calidad en la entrega de servicios. Además, en ambos casos se considera un umbral estático que debe ser alcanzado para iniciar el proceso de escalamiento. Se ha demostrado que, una vez se supera el umbral, el proceso de autoescalamiento puede tardar hasta 10 minutos [31].

Dicho lo anterior, aunque el método de autoescalamiento por violación de umbral es eficaz, puede mejorarse si se combina con técnicas de análisis de tendencias. Por ejemplo, para reducir el periodo de afectación del servicio mientras se ejecuta el escalamiento, se pueden usar técnicas de predicción en series de tiempo que deriven en la toma de acciones correctivas para anticiparse al crecimiento de la demanda. En consecuencia, en este capítulo se presenta la implementación de políticas de autoescalamiento de recursos sobre una infraestructura virtual basada en OpenStack [32], usando técnicas de predicción y un mecanismo de asignación que es ejecutado cuando se superan umbrales estáticos. La infraestructura virtual es implementada sobre hardware real usando la arquitectura propuesta en la Figura **2-1**. Por otra parte, la técnica de predicción usada está basada en HTM [65]. Las entradas para el algoritmo son tomadas directamente de la infraestructura a través de métricas de consumo de recursos, mientras que la automatización del escalamiento se hace mediante un código en Python y haciendo uso de Terraform [39]. Por otro lado, en este capítulo se da cumplimiento a los objetivos específicos 2 y 3 de la investigación, por medio de la definición de políticas de autoescalamiento y de la ejecución de estrategias correctivas, siguiendo recomendaciones SON como se propone en la Figura **1-1**, respectivamente.

4.2. Trabajos relacionados que implementan mecanismos de autoescalamiento en infraestructuras virtuales

El escalamiento es la habilidad que tiene una infraestructura para entregar continuamente un servicio con un desempeño aceptable mientras la carga de trabajo es repartida sobre los recursos eficientemente [95]. Una ventaja de la virtualización de recursos es que desacopla el hardware de sus funciones, habilitando la posibilidad de implementar funciones virtuales de red [92]. En la literatura, son varias las investigaciones que adoptan técnicas de autoescalamiento enfocadas en VNFs (Virtualized Network Function).

Como cualquier recurso, las VNFs pueden escalarse con base en un desempeño deseado. Esta propiedad es considerada en el desarrollo de aplicaciones de red. En [98] los autores proponen un mecanismo de autoescalamiento de VNFs para mantener un desempeño en la red, mientras se minimiza el costo de despliegue. Esta propuesta, meramente analítica, es probada en un escenario de simulación donde se despliega un vEPC dinámicamente. Los resultados muestran que es posible optimizar el costo de operación usando virtualización de recursos. Sin embargo, considerar escenarios donde solo se usan funciones virtualizadas para la operación, se aleja de un enfoque real. Por esta razón, los mismos autores asumen que las redes brindan un mejor desempeño si se combinan soluciones legadas (con hardware de propósito específico) con soluciones virtualizadas. Así, en [99] proponen un algoritmo de escalamiento adaptativo para balancear la relación costo/desempeño en las VNFs desplegadas, mientras se mantiene un nivel de desempeño aceptable en una red móvil. Mediante una simulación en NS-2, se implementan estrategias de autoescalamiento que consideran la operación de equipos legados para salvaguardar costos y tiempo de despliegue. Por otra parte, el problema de escalamiento en redes móviles puede abordarse desglosando los servicios que se prestan en su arquitectura. En [100] los autores proponen un mecanismo de autoescalamiento para las funciones de un MME (Mobility Management Entity) basado en microservicios. Al abstraer las funciones de esta forma, es posible desplegar un ambiente de alta disponibilidad en el que se obtienen mejoras de *throughput* y uso eficiente de recursos. Es importante mencionar que el escalamiento propuesto es horizontal, coordinando la comunicación mediante un balanceador de carga. Los resultados muestran que se pueden disminuir los recursos necesarios para el procesamiento del tráfico de señalización, mientras se mejora el desempeño del MME.

La flexibilidad y escalabilidad que otorga la virtualización, es independiente del tipo de red sobre la que se aplica. Típicamente, los enfoques son aplicados sobre elementos del Core de una red móvil. No obstante, varios autores han desarrollado aplicaciones sobre la RAN (Radio Access Network). En [101] se presenta un servicio *cloud native* para autoorganizar la RAN usando femto celdas SDN y recomendaciones SON [62]. La intención es demostrar la factibilidad de desplegar una WMAN (Wireless Metropolitan Area Network) usando femto celdas que, mediante SDN, comparten información de la red de acceso a un gestor.

La información enriquece un modelo que organiza los recursos compartidos del espectro mientras gestiona la movilidad. Los resultados muestran que con estas celdas es posible resolver conflictos de asignación de recursos en radio. Por otra parte, en [90] se propone la implementación de *Network Slicing* integrando un catálogo de servicios con el despliegue en tiempo real de una RAN virtual, que cumple los requerimientos del servicio. Primero, se implementa toda una infraestructura de nube con múltiples VNFs para la RAN. Luego, se proponen dos escenarios: uno donde se autoescalan los Slices de red para evitar sobre costos y otro donde se maximiza su uso para que los recursos sean utilizados de forma eficiente. Los resultados muestran una mejora significativa en este uso eficiente. Sin embargo, aunque los trabajos sobre la RAN evidencian resultados positivos, tratar de forma independiente la RAN del Core, no garantiza que la optimización de recursos asegure el desempeño esperado de extremo a extremo (E2E). De hecho, en términos de QoE, depende más del tipo de servicio que se consume y de las expectativas del usuario [24, 102].

No obstante, la revisión de literatura evidencia trabajos que consideran el E2E de la red para orquestar servicios usando recomendaciones de la ETSI para NFV [52]. En [103] los autores despliegan un sistema de orquestación escalable E2E para Network Slicing en redes móviles. El sistema está compuesto de varios NFVI (Network Function Virtualization Infrastructure) que son gestionados a través de un orquestador desde el que se implementan los OSS (Operation Suport System) y BSS (Business Suport System). El principal aporte de la investigación, es que demuestra que la escalabilidad de un sistema no está atada a un único dominio de infraestructura. Sin embargo, un escenario multi dominio puede impactar el tiempo de reacción cuando se usan mecanismos de escalamiento reactivos. Por esta razón, al momento de escalar recursos se considera, no solo la asignación eficiente, sino también la reducción del tiempo de respuesta.

Con la adopción de tecnologías de virtualización, un proveedor de infraestructura puede soportar múltiples redes virtuales que garanticen la entrega de múltiples servicios, al tiempo que satisfacen requerimientos de calidad de servicio (QoS, Quality of Service) [104]. En [105] se propone un algoritmo de asignación de recursos denominado RALF (Resource Allocation based on Load Forecasting). Los autores consideran el espacio en caché y los bloques de recursos tempo-frecuenciales para formular un problema de minimización. Luego, usando predicciones de carga para cada red virtual, estiman la probabilidad de sobrecarga de cada servicio y plantean una estrategia de asignación. Los resultados muestran que con RALF se obtiene un efecto positivo en la tasa de uso promedio de recursos y la tasa de pérdida de bits.

Aunque escalar redes garantiza la entrega de múltiples servicios, no sugiere una mejora en la calidad de estos. Por esta razón, algunas investigaciones se preocupan más por el diseño del servicio que por la infraestructura que lo soporta. En [106] se propone una plataforma de código abierto autoescalable para mejorar la entrega de servicios multimedia,

contemplando la integración con componentes de terceros. Basándose en la arquitectura Lambda introducida por Amazon [107], los autores diseñan una plataforma *edge-to-cloud* para añadir valor agregado a la entrega de servicios multimedia para 5G. La plataforma, alineada con las recomendaciones del 5GPPP [108], facilita la integración con herramientas externas para maximizar el impacto positivo en la entrega de servicios. Aunque esté diseño puede facilitar la adopción de tecnologías basadas en nube para la industria multimedia, no garantiza la alta disponibilidad de los servicios debido a que el mecanismo de escalamiento usado está basado en OSM (Open Source MANO) [58]. Por este motivo, el escalamiento es reactivo y basado en umbrales infiriendo que, ante picos de procesamiento, existe un periodo de afectación del servicio dependiente del tiempo de respuesta del orquestador.

Dicho lo anterior, un reto importante en los mecanismos de autoescalamiento es la garantía de alta disponibilidad en la entrega de servicios. No es suficiente con escalar instancias, redes o discos, la comunicación entre los nuevos recursos debe ser coordinada para no afectar la calidad en la entrega. Es común encontrar investigaciones que añaden un elemento adicional a la infraestructura para coordinar los recursos recién añadidos o eliminados por efectos del escalamiento. Por ejemplo, en [109] se investiga cómo pueden unirse un balanceador de carga con las políticas de autoescalamiento de una infraestructura para mejorar la toma de decisiones. En este caso, el balanceador de carga es el elemento adicional que agrega alta disponibilidad en la entrega del servicio, mientras que las políticas de autoescalamiento son efectuadas mediante un framework de orquestación y monitoreo denominado XeniumNFV [110]. Los resultados de la investigación sugieren que los algoritmos de balanceo de carga deben saber a priori la decisión de autoescalamiento para mejorar la QoS percibida y gestionar de forma eficiente los recursos. Este tipo de técnicas, donde se usa un balanceador de carga para garantizar alta disponibilidad, es usado en otras tecnologías como Kubernetes [49]. Sin embargo, Kubernetes es orientado a la gestión de contenedores, pudiendo ser usado para el despliegue de CNFs (Container Network Function). En cuanto a VNFs, las soluciones de nube públicas o privadas se limitan a proveer los recursos de cómputo necesarios para su implementación, pero no sugieren el uso de plataformas o servicios para garantizar alta disponibilidad.

La revisión de literatura evidencia múltiples vías para implementar mecanismos de escalamiento sobre infraestructuras virtuales. Como resultado, hay varios aspectos a considerar que diferencian la implementación entre mecanismos. Por ejemplo, el escalamiento puede ser vertical u horizontal, reactivo o proactivo, de recursos o de servicios, usando criterios asociados a información de la infraestructura o de la calidad percibida por los usuarios, modelando el escalado como una cadena de suministros o un problema de optimización, definiendo políticas dinámicas o estáticas, usando un orquestador, un balanceador de carga o algún elemento externo para coordinar el resultado del escalado. De las posibles combinaciones, en este capítulo se presenta la implementación de una estrategia de autoescalamiento

proactivo de recursos sobre un ambiente virtualizado como el propuesto en la arquitectura de la Figura **2-1**. Los recursos de este ambiente son virtualizados mediante OpenStack. Además, se considera un componente inteligente para la predicción de demanda, basado en el algoritmo HTM previamente descrito en el Capítulo 3. También, se ejecutan las políticas de autoescalamiento a través de herramientas de código abierto como Python y Terraform.

4.3. Escalamiento de recursos

El escalamiento de recursos es la habilidad que tiene una infraestructura o red para entregar continuamente un servicio con calidad aceptable, aun cuando la demanda del mismo varía constantemente [95]. Si el objetivo en el despliegue de una infraestructura, es soportar una demanda creciente de servicios de forma eficiente en el uso de recursos, al tiempo que se desea garantizar la calidad, es necesario evaluar políticas de escalamiento. En la literatura, es común encontrar dos políticas generales: escalar usando hardware y software distribuido, o escalar usando un hardware más potente pero sin distribución [111]. Como resultado, se suelen definir tres métodos de escalamiento principales: escalamiento vertical, escalamiento horizontal y escalamiento elástico [112].

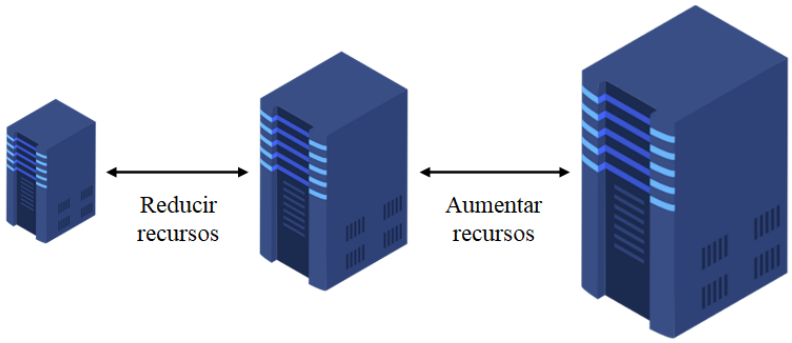
4.3.1. Escalamiento vertical (up/down)

El escalamiento vertical se refiere a la capacidad de ajustar los recursos de una única VM o instancia para hacer frente a cambios en la demanda durante el tiempo de ejecución. En otras palabras, es la capacidad de aumentar (*up*) o reducir (*down*) dinámicamente, recursos como memoria, CPU o almacenamiento en la misma VM. Este método de escalamiento no provoca modificaciones significativas a nivel estructural, lo que lo hace una buena opción si lo que se desea es mantener la simplicidad de una arquitectura en la entrega de un servicio. Sin embargo, aumentar los recursos en función de la capacidad del hardware, infiere que llegará un momento donde habrá una limitación por recursos disponibles para la VM [111, 113]. Además, poseer una única VM para la entrega de un servicio, afecta la alta disponibilidad del mismo si el escalamiento implica reiniciar la instancia. En la Figura 4.1(a) se muestra una abstracción del escalamiento vertical.

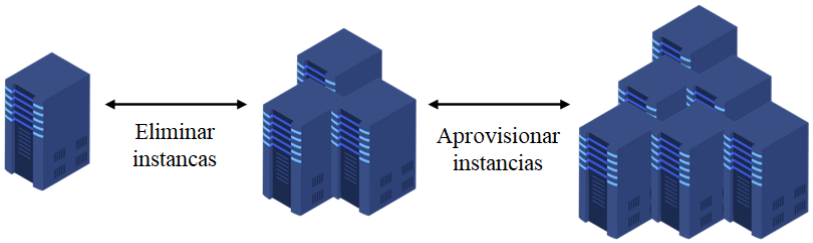
4.3.2. Escalamiento horizontal (out/in)

El escalamiento horizontal se refiere a la capacidad de aprovisionar y lanzar VMs o instancias para hacer frente a cambios en la demanda durante el tiempo de ejecución. Este método de escalamiento se basa en mantener el desempeño en la entrega de un servicio, distribuyendo la carga de trabajo entre múltiples instancias, en lugar de incrementar los recursos de una sola. La principal desventaja de este método es que involucra mayores cambios arquitectónicos que el método anterior debido a que, en lugar de mantener la comunicación con una sola

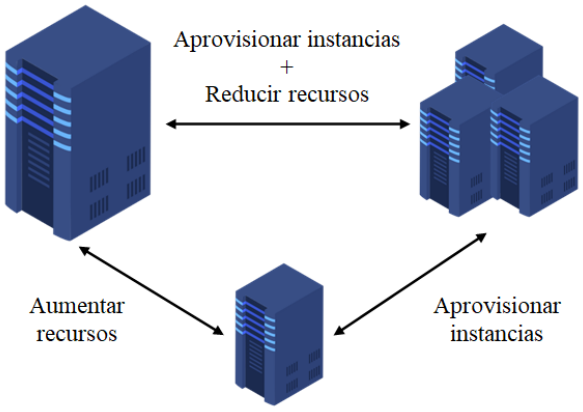
instancia, ahora hay que garantizar la comunicación con y entre varias de ellas [111, 114]. Por esta razón, es común encontrar elementos adicionales en la arquitectura, como balanceadores de carga, cuando se usa este tipo de escalamiento [115]. No obstante, es el método adecuado si se desea garantizar alta disponibilidad en la entrega de un servicio. En la Figura 4.1(b) se muestra una abstracción del escalamiento horizontal.



(a) Escalamiento vertical.



(b) Escalamiento horizontal.



(c) Escalamiento elástico.

Figura 4-1.: Métodos de escalamiento. Adaptado de [111].

4.3.3. Escalamiento elástico

Es la capacidad de escalar en ambas dimensiones, es decir, tanto vertical como horizontalmente. Este método combina los beneficios de los métodos anteriores para adaptar la infraestructura a las variaciones de carga y hacer un uso más eficiente de los recursos. A su vez, la infraestructura debe estar en la capacidad de aumentar o reducir recursos de hardware, así como aprovisionar o eliminar instancias [111]. La principal desventaja de este método es que, si la demanda actual no lleva al límite los recursos de hardware de una única VM, no es claro cuándo se debe aplicar un método u otro. Así mismo, si el objetivo es mantener la alta disponibilidad de un servicio, los métodos elásticos terminan aplicando únicamente un escalamiento horizontal. En consecuencia, la complejidad de un método elástico, está determinada más por sus políticas de escalamiento que por su objetivo de desempeño en la entrega de un servicio. En la Figura 4.1(c) se muestra una abstracción del escalamiento elástico.

En el trabajo realizado en [111] se evidencia una clara preferencia por el escalamiento horizontal cuando se trata de servicios de una red móvil como los que puede brindar un vEPC. La razón es que, en este tipo de servicios, se busca alta disponibilidad en la entrega. No obstante, investigaciones como [116] revelan que un escalamiento vertical puede ser aprovechado en escenarios donde los recursos de cómputo pueden ser escasos. Sin embargo, los autores proponen que, cuando una instancia llega a su límite de demanda de hardware, sea eliminada y creada nuevamente con un *flavor* más grande, lo cual sugiere que el escalamiento no se hace "en caliente", sino más bien, reasigna la entrega del servicio a otra instancia, como se haría en un escalamiento horizontal. Este procedimiento es común en soluciones de nubes públicas y privadas como las que se han mencionado antes en el documento. Por esta razón, considerando que el autoescalamiento infiere el aprovisionamiento o eliminación de nuevas instancias para distribuir la carga, sin perder la disponibilidad del servicio, en esta investigación se prioriza el escalamiento horizontal sobre el vertical para la definición e implementación de las políticas de autoescalamiento.

4.4. Políticas de autoescalamiento que consideran mediciones históricas del uso de recursos obtenidas desde OpenStack

Considerando que las políticas de autoescalamiento deben integrar las predicciones realizadas por el algoritmo HTM en la demanda de recursos, así como el estado actual de los mismos, la definición de umbrales se basa en dos escenarios: uno proactivo, que se anticipa a la demanda de recursos, y uno reactivo, que considera la demanda actual. Se tienen en cuenta las siguientes condiciones:

1. Las decisiones de escalamiento se toman con base en el comportamiento de la CPU y la RAM, y el uso de los recursos de red de una VM como el tráfico en sus interfaces de red.

2. El escalamiento horizontal se prioriza sobre el vertical.

3. Como punto de partida, se implementa una única instancia con un servidor HTTP basado en la librería *apache2* [117]. El tráfico es demandado desde una VM que hace una prueba de estrés al servidor.

4. A medida que el servidor es estresado, usando Apache Bench (`ab`) [118], se mide el tiempo de respuesta ante cambios en la demanda del servicio.

5. La CPU es estresada usando `ab`. Sin embargo, se ha demostrado que `ab` tiene un impacto poco significativo sobre la RAM, por lo que se usa un paquete adicional denominado `stress-ng` para estresar la RAM. Por otro lado, el tráfico en las interfaces de red también varía cuando se usa `ab`.

6. Cuando alguna de las políticas de autoescalamiento se cumple, con Terraform se aprovisiona una nueva instancia y se balancea la carga. No se implementa un balanceador de carga externo.

7. También con Terraform, se eliminan instancias cuando la demanda de recursos decrece a niveles por debajo de los umbrales definidos en la Tabla **4-1**.

8. No se implementa un servicio de almacenamiento persistente para mantener los datos de configuración de las instancias creadas o eliminadas. Esto quiere decir que cualquier tipo de configuración debe ser efectuada durante la creación de la máquina, usando por ejemplo un archivo de *cloud init*, o después mediante un *script*.

9. Se garantiza la entrega continua del servicio. En otras palabras, independientemente de cuál sea la hora de consulta, el servicio siempre estará disponible.

10. Se implementan políticas de escalamiento tanto proactivas como reactivas. Se asume que no se pueden superponer decisiones de escalamiento debido a que se analizan en instantes de tiempo diferentes.

11. Se implementa un escenario donde la demanda del servicio se asemeja a la tendencia de la gráfica mostrada en la Figura **3-12**.

Con base en estas condiciones, en la Tabla **4-1** se definen las variables que determinan la aplicación de una política de autoescalamiento. Las variables abstraen el comportamiento de la infraestructura en valores cuantitativos que, al superar ciertos umbrales, indican que debe aplicarse alguna política.

Tabla 4-1.: Variables a considerar en las políticas de autoescalamiento.

Id.	Variable	Descripción	Unidad	Valor
1	`cpu_usage[t]`	Uso promedio de la CPU en una instancia.	%.	-
2	`ram_usage[t]`	Uso promedio de la RAM en una instancia.	%.	-
3	`thrgpt_usage[t]`	*Throughput* promedio en la NIC de una instancia.	Kbps.	-
4	`pd_cpu_usage`	Predicción del uso promedio de la CPU en una instancia.	%.	-
5	`pd_ram_usage`	Predicción del uso promedio de la RAM en una instancia.	%.	-
6	`pd_thrgpt_usage`	Predicción del *throughput* promedio en la NIC de una instancia.	Kbps.	-
7	`threshold_cpu_max`	Umbral superior para el uso eficiente de CPU en una instancia.	%.	90 %
8	`threshold_cpu_min`	Umbral inferior para el uso eficiente de CPU en una instancia.	%.	10 %
9	`threshold_ram_max`	Umbral superior para el uso eficiente de RAM en una instancia.	%.	90 %
10	`threshold_ram_min`	Umbral inferior para el uso eficiente de RAM en una instancia.	%.	10 %
11	`threshold_thrgpt_max`	Umbral superior para evitar congestión en la NIC de una instancia.	Kbps.	pcl(95)
12	`n_instances`	Número de instancias actualmente desplegadas.	-	-

Donde pcl() representa un percentil medido al conjunto de datos y `t` el instante de tiempo actual sobre el cual se efectúa la medición. En esta misma tabla, se establecen los umbrales máximos y mínimos que determinan las decisiones de escalamiento. Por otro lado, se asume que las variables tienen un comportamiento predecible, esto es, tienen un componente estacionario o una tendencia lineal en su comportamiento. Lo anterior es importante en los casos donde se presenta una anomalía. Como se demuestra en el Capítulo 3, HTM puede aprender las anomalías aun cuando no hacen parte del comportamiento normal. Sin embargo, las anomalías no deben inferir en la aplicación de una política de autoescalamiento cuando son efímeras. Esta consideración es tenida en cuenta durante la aplicación de las políticas de autoescalamiento.

Entre tanto, las variables 1, 2 y 3 de la Tabla **4-1**, se refieren a mediciones promedio por el hecho de ser agregadas a través del tiempo. Esto es, periódicamente se agregan los valores de una ventana de tiempo para construir la secuencia sobre la cual se realiza el monitoreo. Por efectos de la implementación, la agregación se hace cada 30 segundos mientras que el monitoreo se hace casi en tiempo real, por lo que el escalamiento responde a condiciones que el sistema presenta en una ventana de tiempo próxima al instante actual.

4.4.1. Algoritmo para la implementación de políticas de escalamiento

Las variables definidas en la Tabla **4-1** son consideradas en la implementación de las políticas de autoescalamiento, a través de un algoritmo abstraído en el diagrama de flujo de la Figura **4-2**. En el Anexo E se incluye el pseudocódigo que facilita la implementación de este algoritmo, así como el detalle de cada uno de sus estados. Por otro lado, la implementación del ambiente real, incluyendo los componentes de la arquitectura presentada en la Figura **2-1**, HTM y el código para aplicar las políticas de autoescalamiento, puede consultarse en [119]. Este código es, en esencia, quien permite la interacción entre las herramientas de la arquitectura para hacer efectivo el autoescalamiento propuesto.

4.5. Experimentación

El punto de partida es una única instancia con un servidor HTTP basado en la librería *apache2*. El despliegue de esta instancia es automatizado con Terraform usando el código disponible en el Anexo D. Luego del despliegue, se aprecia que el paquete de exportación de métricas `prometheus-node-exporter` está activo, como se ve usando el comando `ps aux | grep prome`.

```
1  $ ps aux | grep prome
2  prometh+  2328  0.4  0.5 558784 23168 ?        Ssl  Feb20  23:11
```

Por otro lado, con Grafana se visualiza el estado en cuanto a consumo de recursos de la instancia. Cuando la instancia es estresada usando `ab`, se logra una variación en el uso de CPU y ancho de banda de la instancia, como se aprecia en las gráficas *CPU Basic* y *Network Traffic Basic* de la Figura **4-3**. Lo que permite esta variación es que `ab` genera solicitudes HTTP masivas al servidor, razón por la cual, la CPU y el tráfico de red es alterado. Sin embargo, la RAM no se ve afectada. Para estresarla, es necesario otro procedimiento que, típicamente, es de lectura/escritura de archivos o cálculos de potencias en periodos de tiempo fijos. En este caso se usa el paquete `stress-ng` para tal propósito. El resultado son picos idénticos a los obtenidos en el consumo de CPU y tráfico de red, como se observa en el gráfico *Memory Basic* de la Figura **4-3**. Para lograr el efecto de periodicidad que se observa en la demanda de recursos, se usa un código en Python que puede ser consultado en [119].

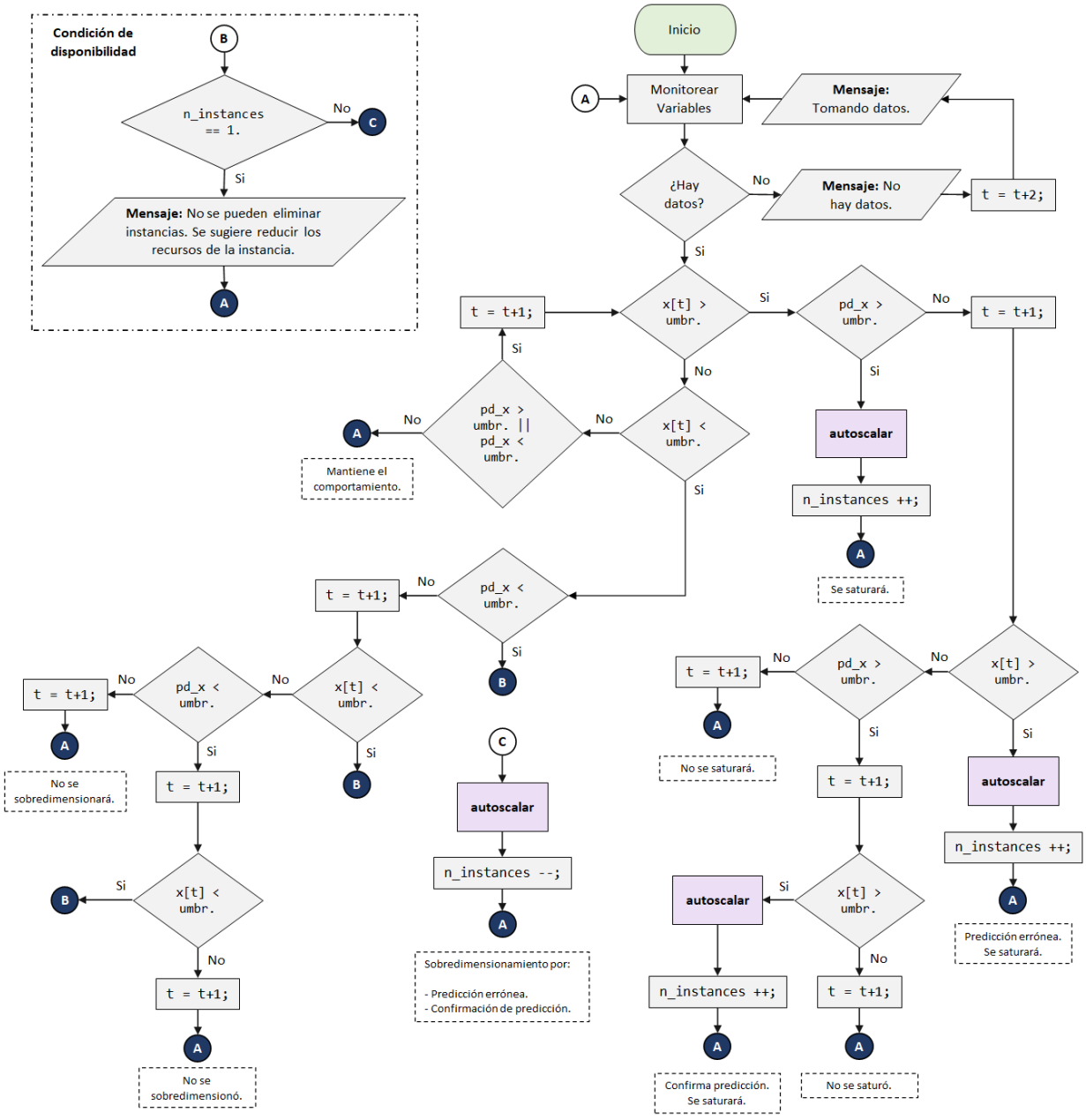


Figura 4-2.: Diagrama de flujo para la implementación de las políticas de autoescalamiento.

Combinando la demanda de recursos provocada por `ab` y `stress-ng` de forma periódica, es posible simular un escenario que se aproxime a una demanda real. Como se dijo, la forma periódica es obtenida siguiendo la tendencia de la demanda real presentada en la Figura **3-12**. De este modo, la simulación logra generar un comportamiento estacionario pero en un periodo de simulación del orden de minutos, en lugar de días. Por otra parte, hay que mencionar que el algoritmo de autoescalamiento es aplicado sobre este comportamiento.

Figura 4-3.: Comportamiento de los recursos del servidor cuando es estresado usando `ab` y `stress-ng`.

4.5.1. Resultados de las políticas de autoescalamiento propuestas

En la práctica, antes de aplicar las políticas de autoescalamiento, es necesario construir el algoritmo de HTM para que aprenda las tendencias del comportamiento de los recursos. En [119] se pueden consultar los parámetros con los que HTM es entrenado. Una vez HTM es entrenado, se inicia el algoritmo propuesto en la Figura **4-2**. Como primer resultado se evalúa la capacidad del algoritmo para escalar recursos. En la Figura **4-4** se muestra qué pasa con los recursos cuando se efectúa el escalamiento.

En la primera etapa, se genera una demanda que supera los umbrales máximos de consumo establecidos. Esta etapa es crucial para que HTM pueda aprender cuál es la tendencia y mejore su precisión en la predicción. Las estadísticas de respuesta del servicio reportadas por `ab` en esta etapa, son las que se presentan en la Figura **4-5**. De las estadísticas presentadas, son importantes para esta investigación, el tiempo total de conexión (medido en milisegundos) y los percentiles de la medición de latencia en el total de solicitudes realizadas.

Como se observa en la Figura **4-5**, en total se hicieron 37419 solicitudes en un periodo de tiempo de aproximadamente 70 segundos. El tiempo promedio de conexión fue de 626 ms con una desviación estándar de 4132 ms, una mediana de 77 ms y un tiempo máximo de respuesta de 58919 ms. Así mismo, cerca del 80 % de las solicitudes tardaron menos de 90 ms en responderse. Esta información sirve como punto de partida para cuando se aplica el escalamiento de recursos.

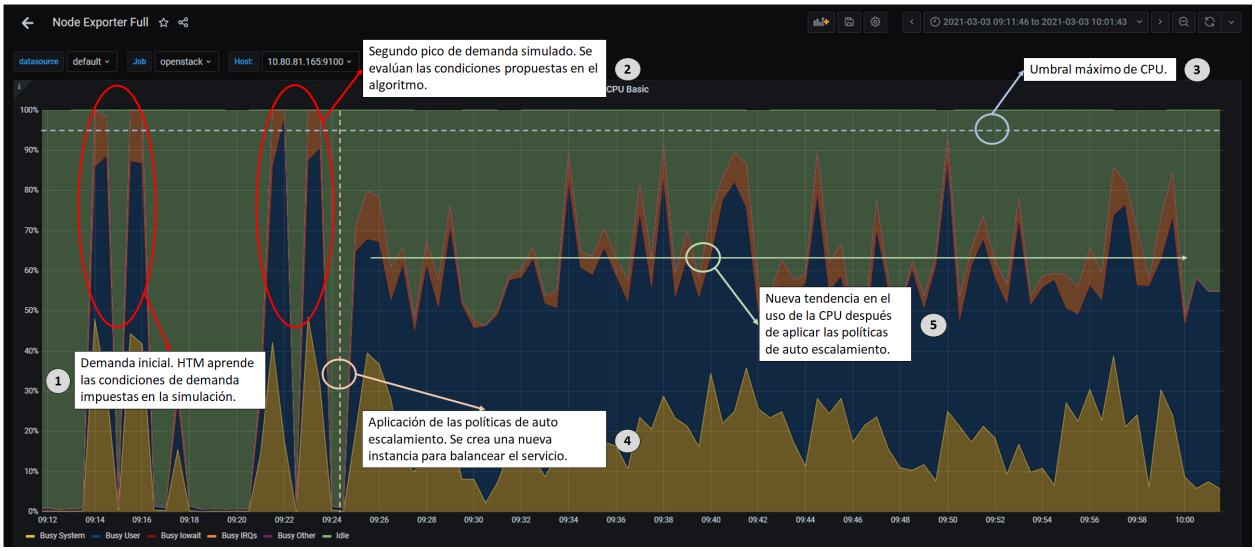


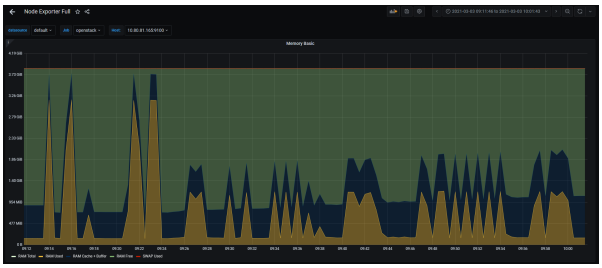
Figura 4-4.: Escalamiento de recursos efectuado automáticamente por el algoritmo propuesto.

Continuando con las etapas de la Figura **4-4**, la segunda consiste en generar un comportamiento en la demanda de recursos que pueda llevar al escenario a requerir un escalamiento. Aquí, se evalúan cuáles son las condiciones actuales del servidor. En la misma figura, se observa que antes de que los recursos del servidor lleguen a un nuevo pico de uso y superen el umbral (tercera etapa), se aplican las políticas de autoescalamiento para crear una nueva instancia que balancee el servicio. La creación de una nueva instancia compete la cuarta

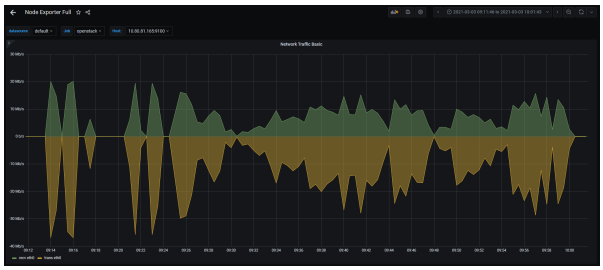
```
Server Software:        Apache/2.4.38
Server Hostname:        10.80.81.165
Server Port:            80

Document Path:          /
Document Length:        154 bytes

Concurrency Level:      1000
Time taken for tests:   69.934 seconds
Complete requests:      37419
Failed requests:        174
   (Connect: 0, Receive: 0, Length: 174, Exceptions: 0)
Total transferred:      15793576 bytes
HTML transferred:       5736346 bytes
Requests per second:    535.06 [#/sec] (mean)
Time per request:       1868.933 [ms] (mean)
Time per request:       1.869 [ms] (mean, across all concurrent requests)
Transfer rate:          220.54 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0  226 728.2     37    7195
Processing:     0  400 4078.1     37   58896
Waiting:        0  163 2149.6     36   58896
Total:          1  626 4132.8     77   58919

Percentage of the requests served within a certain time (ms)
  50%     77
  66%     80
  75%     84
  80%     88
  90%   1085
  95%   1106
  98%   3127
  99%   7491
 100%  58919 (longest request)
```

Figura 4-5.: Estadísticas reportadas por Apache Bench antes del escalamiento.

etapa y permite distribuir las solicitudes del servicio para evitar escenarios de congestión. Como consecuencia, el servidor logra alcanzar un comportamiento estable, por debajo del máximo umbral propuesto para el consumo de CPU, como se visualiza en la quinta etapa. La Figura 4.6(a) muestra el comportamiento de la RAM en el mismo periodo de tiempo, mientras que la Figura 4.6(b) muestra el comportamiento del tráfico de red. Si bien las tres variables son monitoreadas de forma simultánea, el disparador de escalamiento, en este caso, fue generado por la demanda de uso de la CPU.



(a) Comportamiento de la RAM.



(b) Comportamiento del tráfico de red.

Figura 4-6.: Comportamiento de los recursos del servidor durante el escalamiento.

Una vez los recursos se estabilizan, las estadísticas reportadas por `ab` muestran una mejoría en la entrega del servicio, como se muestra en la Figura **4-7**. En este caso, se hicieron 296358 solicitudes en un periodo de tiempo de aproximadamente 70 segundos. El tiempo promedio de conexión fue de 168 ms con una desviación estándar de 423 ms, una mediana de 78 ms y un tiempo máximo de respuesta de 20094 ms. Así mismo, cerca del 80 % de las solicitudes tardaron menos de 85 ms en responderse. En general, las estadísticas reportadas muestran una mejoría en el comportamiento de los recursos respecto al momento antes del escalamiento.

Por otra parte, el tiempo de ejecución del algoritmo puede dividirse en dos etapas. Una primera que contempla el tiempo de construcción de HTM y otra que considera cuánto tarda en efectuarse un escalamiento una vez se cumple alguna de las condiciones. La primera etapa ronda en promedio los 3 minutos y solo debe efectuarse una vez antes de iniciar el monitoreo de recursos, o cada vez que se modifique un parámetro de HTM. La segunda etapa, por otro lado, puede verse desde varios puntos de vista. En los experimentos realizados, desde que se detecta la necesidad de escalamiento hasta efectuarlo, pasa en promedio un poco más de 2 minutos cuando es reactivo. No obstante, si el escalamiento es proactivo, dependiendo de la cantidad de ventanas de tiempo predichas, se reduce el tiempo de ejecución. Dado que, en este caso, la granularidad temporal es de 30 segundos, el tiempo de escalamiento se reduce en este valor con una sola ventana de predicción. Independientemente del método que efectúe el algoritmo (reactivo o proactivo), la arquitectura propuesta demuestra que se pueden efectuar

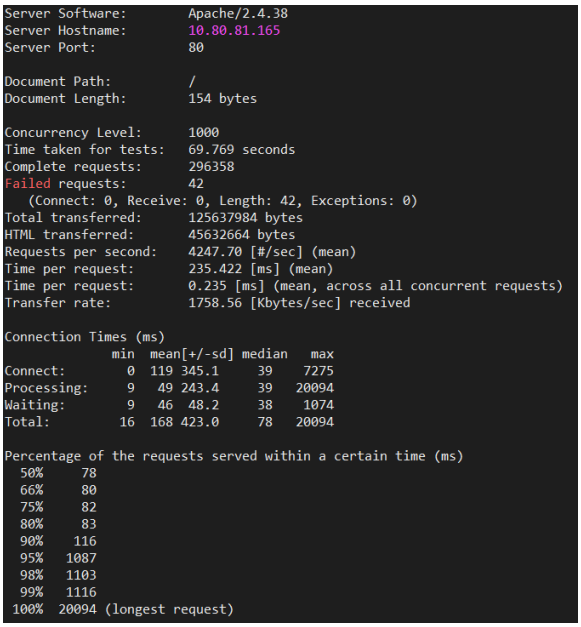


Figura 4-7.: Estadísticas reportadas por Apache Bench después del escalamiento.

políticas de autoescalamiento en el orden de un poco más de 2 minutos como máximo. En comparación con mecanismos de autoescalamiento de nubes públicas, es un resultado que reduce el tiempo de ejecución a más de la mitad. Sin embargo, es importante mencionar que las nubes públicas, al momento de autoescalar VMs, consideran componentes como balanceadores de carga externos para optimizar la distribución de solicitudes al servicio, introducen conceptos de alta disponibilidad que van, desde disponibilidad de puertos físicos de una tarjeta de red, hasta disponibilidad geográfica, etc.

4.6. Discusión

En este capítulo se discutió acerca de la ejecución de políticas de autoescalamiento sobre el escenario de experimentación propuesto en la Figura **2-1**. Las políticas de autoescalamiento fueron abstraídas mediante un algoritmo que considera dos formas de asignación, una reactiva tomando el estado actual de los recursos, y otra proactiva que considera la predicción realizada por el algoritmo HTM para estimar cuál será el comportamiento de los recursos en la siguiente ventana de tiempo. A través de la implementación del algoritmo propuesto, se pudo autoescalar recursos, identificando varias etapas en el procedimiento. Inicialmente, HTM aprende la tendencia en el comportamiento de recursos, luego, considerando el estado actual y las predicciones, el algoritmo de autoescalamiento evalúa si existe una violación de los umbrales propuestos. Si esta condición se cumple, se efectúa el escalamiento, distribuyendo las solicitudes del servicio entre las nuevas instancias creadas.

Los resultados muestran que es posible mejorar el tiempo de respuesta del servidor una vez se efectúa el escalamiento descrito por el algoritmo. Así mismo, se pudo evidenciar de manera gráfica, cómo el escalamiento lleva al servidor, de un estado de saturación, a un comportamiento dentro de los límites de los umbrales establecidos para el comportamiento normal. Adicionalmente, en comparación con soluciones de autoescalamiento de nubes públicas, el tiempo transcurrido desde que se detecta la necesidad de escalar, hasta que se escala, se reduce a más de la mitad. De igual manera, la implementación es realizada usando herramientas de código abierto en un ambiente relativamente asequible, facilitando la adopción de esta solución de autoescalamiento dada su flexibilidad y fácil réplica en otros ambientes. Además, en comparación con soluciones como OSM, hay una mejora de la capacidad de escalamiento, permitiendo evolucionar de un escalamiento reactivo, basado en umbrales estáticos, a uno proactivo como el propuesto.

5. Conclusiones y trabajos futuros

En este capítulo se discuten las principales conclusiones obtenidas de esta investigación. Además, se discuten cuáles fueron los principales inconvenientes durante su elaboración y se proponen trabajos futuros.

5.1. Conclusiones

- De la revisión de literatura sobre arquitecturas, ambientes y escenarios de experimentación para la aplicación de políticas de autoescalamiento, se pudo identificar que, aunque la mayoría de las investigaciones mencionan qué y cuántos recursos de hardware y software se usan para sus experimentos de autoescalamiento, no es claro cuál debería ser el procedimiento para poder replicar un escenario de características similares. Por esta razón, se propuso un escenario de experimentación asequible para computación en la nube fácilmente implementable en ambientes locales. En este se describen, tanto los recursos de hardware y software usados, como el procedimiento llevado a cabo para su implementación. Todo el escenario está constituido por tecnologías de código abierto y usa componentes cuya instalación puede hacerse en pocos pasos. No obstante, podría limitar los recursos de las aplicaciones que se despliegan, a la misma cantidad de recursos usados para acondicionar el ambiente de experimentación. Para evitar este inconveniente, el escenario es fácilmente configurable para ambientes con mayor disponibilidad de recursos de cómputo, almacenamiento y red.

- En la revisión de literatura enfocada a encontrar cuáles son las características de los métodos más usados de ML, en el contexto de redes de telecomunicaciones, para predecir la demanda de un servicio o la carga de un elemento de red, se identificó que, en la mayoría de casos, se requiere un conjunto de datos previos para el entrenamiento de un modelo predictivo, lo cual lo vuelve vulnerable al cambio de contexto. Por esta razón, se propone el uso de un algoritmo denominado HTM, el cual incluye un componente de aprendizaje por refuerzo que le permite adaptarse a estos cambios. Además, HTM está en la capacidad de detectar anomalías y predecir con precisión el comportamiento de una tendencia. Para demostrar la eficacia del algoritmo, se aplicó sobre una serie de tiempo, correspondiente a datos de una red móvil real, donde se incluyeron irregularidades de forma intencional. Los resultados muestran que HTM minimiza el RMSE entre la predicción y el valor que en realidad toma la serie de tiempo. Además,

se demuestra que las anomalías pueden ser detectadas. También, se pudo evidenciar que HTM considera algunas anomalías como parte del comportamiento de la serie y las predice aun cuando no suceden. Por esta razón, asumiendo la aplicación del algoritmo sobre series de tiempo que se comportan de forma estable, HTM es una opción adecuada para predecir tendencias en el contexto de esta investigación.

- Se revisaron investigaciones que usan algún tipo de mecanismo de autoescalamiento para servicios o aplicaciones en redes de telecomunicaciones sobre infraestructuras virtuales, destacando soluciones para redes móviles LTE o enfocadas a 5G. De la revisión, se identificaron múltiples vías para implementar un mecanismo de autoescalamiento, que van desde un escalamiento vertical sobre elementos de gestión de la red como un vMME, hasta cadenas de suministros para maximizar la disponibilidad de un servicio a través de VNFs. Considerando la existencia de estas vías, se propuso un mecanismo de autoescalamiento que se adapta al escenario de experimentación propuesto. En el mecanismo, se usa HTM y umbrales estáticos para disparar el escalamiento ya sea proactivo o reactivo.

- El mecanismo de autoescalamiento propuesto fue abstraído en un algoritmo que considera, tanto un escenario reactivo, como uno proactivo. Considerando este algoritmo y el escenario de experimentación propuesto, se ejecutaron pruebas de estrés para llevar al límite las mediciones de CPU, RAM y volumen de tráfico sobre las interfaces de red de un servidor HTTP. Luego, usando el mecanismo de autoescalamiento propuesto, se demostró que es posible distribuir la carga del servicio sobre instancias recién creadas, liberando recursos del servidor inicialmente estresado. También, los resultados muestran que es posible mejorar la latencia de la conexión en la entrega del servicio. Por otro lado, en comparación con soluciones de autoescalamiento de plataformas de computación en la nube conocidas, el mecanismo propuesto es capaz de escalar recursos en un tiempo más corto. Es importante mencionar que toda la implementación fue realizada usando herramientas de código abierto.

5.2. Trabajos futuros

Son varios los trabajos futuros que se derivan de esta tesis, entre ellos:

- Evaluar cómo se pueden mejorar las condiciones de alta disponibilidad de un servicio a través de mecanismos de autoescalamiento proactivos que consideren elementos externos como balanceadores de carga, servicios DNS, DHCP (Dynamic Host Configuration Protocol) o elementos de red como un *Proxy*.

- Integrar el mecanismo de autoescalamiento propuesto con orquestadores de red como OSM u ONAP. Como se mencionó, Terraform es una solución IaC, por lo que un

orquestador podría brindar funcionalidades adicionales como un catálogo de servicios o la posibilidad de implementar la orquestación de *Network Slicing*.

- Considerar la integración del mecanismo de autoescalamiento con tráfico proveniente de una red real, para evaluar su eficacia en la asignación de recursos sobre una infraestructura que soporte servicios a usuarios finales.

- Utilizar el mecanismo de autoescalamiento propuesto para mantener la alta disponibilidad de otro tipo de servicios como *videostreaming*, *live videostreaming* o servicios asociados a IoT. Además, evaluar cómo puede usarse este mecanismo para optimizar la asignación de recursos asignados a los servicios soportados en un *Network Slice*.

- Integrar múltiples VIM en el mismo escenario de experimentación para extender la disponibilidad de recursos entre varios gestores de infraestructura simulando así múltiples *Datacenters*. En este escenario extendido, si se da un escalamiento horizontal usando gestores distintos, existe un reto adicional para garantizar la conectividad.

A. Anexo: NFV y SDN

La virtualización de funciones de red (NFV, *Network Function Virtualization*) es una tecnología que permite transformar dispositivos de red físicos, de propósito específico, en recursos virtuales de propósito general. Por lo tanto, las funciones de red se independizan del hardware que las ejecuta [91]. Vale resaltar que el ETSI (European Telecommunications Standards Institute) estandariza la solución NFV en el año 2012 proponiendo la arquitectura mostrada en la Figura **A-1** [92].

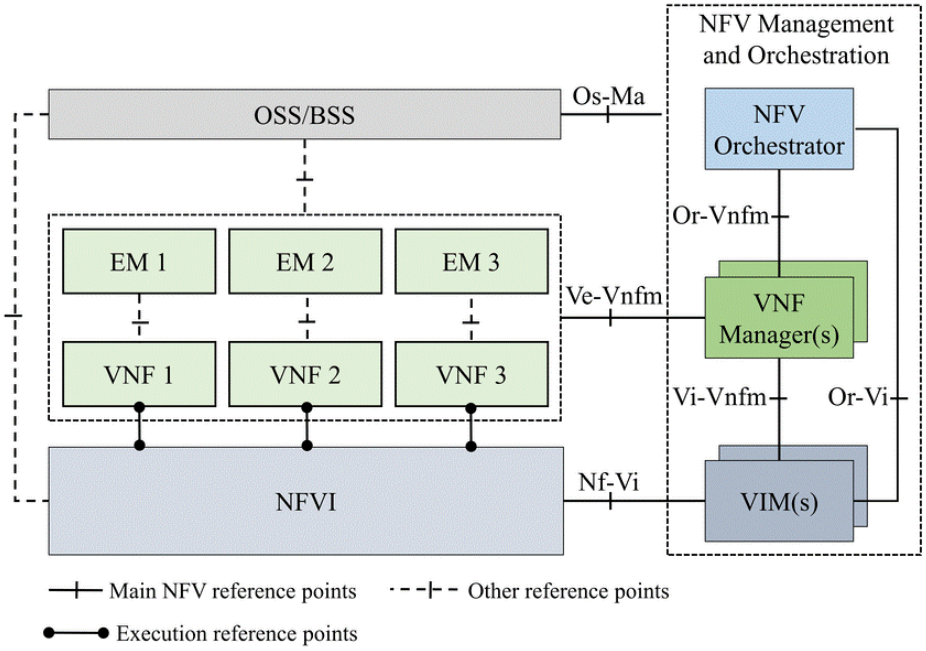


Figura A-1.: Arquitectura NFV propuesta por la ETSI [92].

Por otra parte, las redes definidas por software (SDN, *Software Defined Networking*) son un paradigma de arquitectura de redes en el que se abstrae el plano de control, que posee la lógica de control de flujos en la comunicación, del plano de datos, donde se encuentra el tráfico de datos. Esta idea fue concebida en el proyecto Clean Slate financiado por el programa GENI (Global Environment for Networking Innovations) en 2006. No obstante, pasaron dos años para que investigadores de la Universidad de Stanford implementaran una arquitectura de red usando un protocolo de código abierto para el control de flujos entre planos denominado OpenFlow [120]. La arquitectura resultante se denominó SDN y se muestra en la Figura **A-2**.

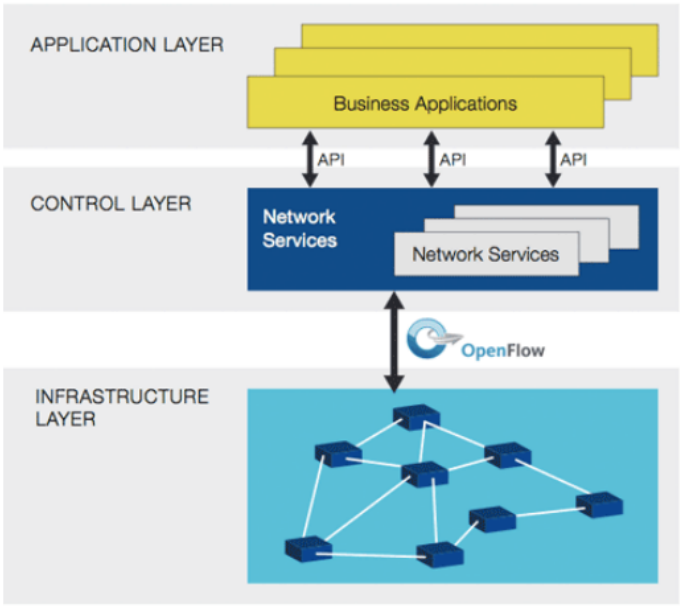


Figura A-2.: Arquitectura SDN [120].

La arquitectura NFV de la Figura **A-1** se acopla con la arquitectura SDN de la Figura **A-2** en la capa de gestión de infraestructura virtual (VIM, *Virtual Infrastructure Management*). Como resultado, y de manera muy general, los recursos virtuales se interconectan entre sí usando SDN. Por lo tanto, la implementación de SDN queda en el dominio de la VIM o, en el caso de redes más extensas, en el dominio multi-VIM. Existen soluciones ampliamente conocidas que internamente poseen una VIM, tales como AWS (Amazon Web Service), GCP (Google Cloud Platform) o Microsoft Azure, todas ellas comerciales. A su vez, algunas soluciones de código abierto son OpenVIM, OpenNebula u OpenStack. OpenStack es un software que controla recursos de cómputo, almacenamiento y lógica de red, haciendo las veces de un *datacenter* liviano que puede ser instalado en una única VM (Virtual Machine). Por otro lado, se encuentra en constante crecimiento gracias a los desarrollos de la comunidad examinados por The Linux Foundation [1]. Sin importar el tipo de solución elegida para la VIM, todas están habilitadas para implementar dinamismo, flexibilidad y elasticidad de recursos en la infraestructura de red.

B. Anexo: Despliegue de MicroStack

MicroStack es una distribución de OpenStack que corre sobre un único paquete de Snap. Fue desarrollada por Canonical para propósitos experimentales y ha evolucionado para ejecutarse como plataforma de *Edge Computing* para IoT (Internet of Things). Sobre MicroStack corren todos los componentes de OpenStack y puede instalarse de forma sencilla. Los requerimientos mínimos de hardware recomendados por Canonical son 2 CPUs, 8 GB de memoria y al menos 100 GB de disco. Dependiendo del uso que se quiera dar, se necesita al menos 1 NIC de 100 Mbps. Usando un OS Ubuntu 20.04 LTS, con 4 CPU, 8 GB de memoria, 200 GB de disco y 1 NIC de 1 Gbps, se ejecutan los siguientes comandos para su instalación:

```
sudo apt update && sudo apt upgrade
sudo apt install snapd
sudo snap install microstack --devmode --beta
```

Cuando el proceso de instalación finaliza, en el terminal debería aparecer un mensaje como el siguiente:

```
microstack (beta) stein from Canonical installed
```

Luego, debe inicializarse la aplicación usando el siguiente comando:

```
sudo microstack init --auto --control
```

Con este, se configuran las redes y bases de datos necesarias para el funcionamiento de MicroStack. Puede tardar, en el hardware recomendado, de 20 a 25 minutos en ejecutarse. A la fecha de la escritura del presente documento, la última versión disponible de OpenStack en MicroStack es Stein. Una vez inicializa la aplicación, se interactúa accediendo a:

```
http://localhost:80/
```

El usuario por defecto es `admin` y la contraseña se obtiene ejecutando `sudo snap get microstack config.credentials.keystone-password`. Si todo está bien, en pantalla se visualiza una interfaz como la de la Figura 3.

C. Anexo: Despliegue de Prometheus y Grafana

Usando el mismo sistema operativo y dimensionamiento que MicroStack (ver el Anexo B), el despliegue de Prometheus y Grafana se hace ejecutando los siguientes comandos. Antes, es necesario instalar Docker para poder hacer el despliegue de los contenedores. Para esto se pueden usar los siguientes comandos:

```
sudo apt-get remove docker docker-engine docker.io containerd runc
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg-agent
software-properties-common && sudo apt install net-tools
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
# Para validar la llave
sudo apt-key fingerprint 0EBFCD88
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/
ubuntu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io
```

A continuación, se procede a descargar las imágenes de los contenedores. Es importante mencionar que los contenedores corren sobre el Kernel del OS que los hospeda, razón por la cual son eficientes en la demanda de recursos como vCPU y vRAM pero no necesariamente en disco, debido principalmente a la escritura de *logs*. Los comandos para descargar Prometheus son:

```
sudo docker pull prom/Prometheus
```

Cuando la imagen del contenedor descarga correctamente, se configura un archivo `prometheus.yml` para extraer las métricas de OpenStack. El archivo debe estar en la carpeta `/etc/prometheus` del host y debe contener la siguiente información. Si la carpeta no existe puede crearse usando el comando sudo `mkdir -p /etc/prometheus`:

```
global:
  scrape_interval:     15s
```

```
    evaluation_interval: 15s

  scrape_configs:
    - job_name: 'openstack'
      openstack_sd_configs:
        - role: 'instance'
          region: 'RegionOne'
          identity_endpoint: 'http://localhost:5000/v3/'
          username: '<username>'
          domain_id: 'default'
          project_name: '<project_name>'
          password: '<user_password>'
      relabel_configs:
        - source_labels: [__meta_openstack_public_ip]
          target_label: __address__
          replacement: '$1:9100'
        - source_labels: [__meta_openstack_tag_prometheus]
          regex: true.*
          action: keep
        - source_labels: [__meta_openstack_tag_node_exporter]
          regex: true.*
          action: keep
        - action: labelmap
          regex: __meta_openstack_(.+)
```

Los campos de nombre de usuario, nombre de proyecto y contraseña, se deben reemplazar por los valores de la configuración de OpenStack. Por último, el contenedor se ejecuta usando el siguiente comando:

```
sudo docker run -d --name=prometheus -p 9090:9090 -v /etc/prometheus/
prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus
```

El despliegue correcto de Prometheus puede validarse accediendo a la URL `http://localhost:9090/graph` y luego, en la pestaña de `Status`, `Service Discovery` deben visualizarse las métricas de los recursos virtuales para cada instancia en OpenStack que posea el paquete `prometheus-node-exporter` al momento de su despliegue. Para Grafana el despliegue es más simple. El contenedor únicamente es halado y lanzado con un *port-forwarding* al 3000, donde se despliega la interfaz gráfica. Los comandos son los siguientes:

```
sudo docker pull grafana/grafana
sudo docker run -d --name=grafana -p 3000:3000 grafana/grafana
```

El siguiente paso es configurar Grafana para que se conecte con Prometheus y puedan visualizarse las métricas de recursos virtuales. Grafana cuenta con un repositorio de plantillas donde se incluye un dashboard para monitorear métricas de OpenStack [57]. El dashboard trae preconfigurado todos los parámetros necesarios para la conexión, configuración y visualización de métricas de Prometheus. El resultado debe ser un dashboard idéntico al de la Figura **2-4**.

D. Anexo: Despliegue de Terraform

Para integrar Terrafrom con OpenStack es necesario contar con las imágenes de OS (preferiblemente en formato QCOW2) previamente cargadas en *Nova*, las llaves de seguridad (`application_credential_secret`) y las credenciales que fueron usadas antes de la integración. Una vez esto es garantizado, se procede a descargar y configurar Terraform.

```
# Al momento de escribir el documento, la version de Terraform es 0.13.4
cd && wget https://releases.hashicorp.com/terraform/0.13.4/terraform_0.13.4
_linux_amd64.zip
unzip terraform*.zip
sudo rm terraform*.zip
sudo mv terraform /usr/local/bin/
```

Para comprobar la instalación se ejecuta `terraform version` en el terminal. Antes de inicializar Terraform, es necesario crear una carpeta para incluir sus archivos de configuración. Puede hacerse mediante los siguientes comandos:

```
mkdir ~/terraform && cd ~/terraform
mkdir openstack
```

Luego, hay que hacer una configuración previa para que Terraform se integre con OpenStack. Prrimero, es necesario asignar las variables de entorno referentes a la API de OpenStack. Así, si el OS usado es Ubuntu 20.04 LTS, debe añadirse al archivo `/etc/environment` la siguiente información:

```
OS_AUTH_URL=http://localhost:5000/v3/
OS_PROJECT_ID=<project_id>
OS_PROJECT_NAME="<project_name>"
OS_USER_DOMAIN_NAME="Default"
OS_PROJECT_DOMAIN_ID="default"
OS_USERNAME="<user_name>"
OS_PASSWORD=<password>
OS_REGION_NAME="RegionOne"
OS_INTERFACE=public
OS_IDENTITY_API_VERSION=3
```

La misma información debe añadirse al archivo `~/.bashrc` con el comando para exportar las variables como se muestra a continuación:

```
export OS_AUTH_URL=http://localhost:5000/v3/
export OS_PROJECT_ID=<project_id>
export OS_PROJECT_NAME="<project_name>"
export OS_USER_DOMAIN_NAME="Default"
export OS_PROJECT_DOMAIN_ID="default"
export OS_USERNAME="<user_name>"
export OS_PASSWORD=<password>
export OS_REGION_NAME="RegionOne"
export OS_INTERFACE=public
export OS_IDENTITY_API_VERSION=3
```

Luego, se debe ejecutar `source ~/.bashrc` para hacer efectivo el cambio. Siguiente a esto, en la carpeta `openstack` creada antes, se crea un archivo `instance.tf` donde se define quién es el proveedor de infraestructura para Terraform y los parámetros necesarios para la creación de todos los recursos de infraestructura necesarios para el despliegue de una instancia. En el caso de OpenStack, los recursos pueden ser la imagen base de la instancia, el *flavor*, grupos de seguridad, IPs flotantes y configuración adicional mediante un archivo de *cloud init*. Luego, desde Terraform, es posible gestionar la infraestructura para la creación de la instancia. En este caso, el archivo `instance.tf` debe contener lo siguiente:

```
provider "openstack" {
}

data "openstack_images_image_v2" "debian-buster" {
  name        = "debian10"
  most_recent = true
}

data "openstack_compute_flavor_v2" "m1-micro" {
  name  = "<flavor_name>"
}

resource "openstack_compute_instance_v2" "debian-buster" {
  name            = "debian-buster"
  image_id        = data.openstack_images_image_v2.debian-buster.id
  flavor_id       = data.openstack_compute_flavor_v2.m1-micro.id
  security_groups = [
    openstack_networking_secgroup_v2.buster.name
  ]
  user_data       = data.template_file.debian.template
```

```
21
22      metadata = {
23        prometheus = "true"
24        node_exporter = "true"
25      }
26
27      network {
28        name = "<network_name>"
29      }
30    }
31
32    data "openstack_networking_floatingip_v2" "debian-buster-fip" {
33        address  = "<available_floating_ip>"
34    }
35
36
37    resource "openstack_compute_floatingip_associate_v2" "debian-buster-fip" {
38        floating_ip = data.openstack_networking_floatingip_v2.debian-buster-fip.
    address
39        instance_id = openstack_compute_instance_v2.debian-buster.id
40    }
41
42    resource "openstack_networking_secgroup_v2" "buster" {
43        name        = "buster"
44        description = "Buster Security Group"
45    }
46
47    resource "openstack_networking_secgroup_rule_v2" "node_exporter" {
48      direction         = "ingress"
49      ethertype         = "IPv4"
50      protocol          = "tcp"
51      port_range_min    = 9100
52      port_range_max    = 9100
53      remote_ip_prefix  = "0.0.0.0/0"
54      security_group_id = openstack_networking_secgroup_v2.buster.id
55    }
56
57    resource "openstack_networking_secgroup_rule_v2" "ssh" {
58      direction         = "ingress"
59      ethertype         = "IPv4"
60      protocol          = "tcp"
61      port_range_min    = 22
```

```
    port_range_max    = 22
    remote_ip_prefix  = "0.0.0.0/0"
    security_group_id = openstack_networking_secgroup_v2.buster.id
  }

  resource "openstack_networking_secgroup_rule_v2" "http" {
    direction         = "ingress"
    ethertype         = "IPv4"
    protocol          = "tcp"
    port_range_min    = 80
    port_range_max    = 80
    remote_ip_prefix  = "0.0.0.0/0"
    security_group_id = openstack_networking_secgroup_v2.buster.id
  }

  resource "openstack_networking_secgroup_rule_v2" "icmp_v4" {
    direction         = "ingress"
    ethertype         = "IPv4"
    protocol          = "icmp"
    remote_ip_prefix  = "0.0.0.0/0"
    security_group_id = openstack_networking_secgroup_v2.buster.id
  }
```

En resumen, el archivo asume que existe una imagen de OS en OpenStack con el nombre `debian-buster`. Luego, define el tamaño de la máquina en términos de vCPU, vRAM y almacenamiento. Seguido, se crea una contraseña SSH para su acceso seguro. Aquí es importante considerar el archivo de *cloud init* previamente discutido en el Capítulo 2 y pasarlo como plantilla para que Terrafrom cree la VM considerando su contenido. Para esto se crea un archivo `user_data.tf` en la carpeta `~/terraform/openstack` con el siguiente contenido:

```
  data "template_file" "debian" {
    template = "${file("$HOME/terraform/files/debian.tpl")}"
  }

  data "template_cloudinit_config" "debian" {
    gzip          = false
    base64_encode = false

    part {
      filename     = "init.cfg"
      content_type = "text/cloud-config"
```

```
12        content      = data.template_file.debian.rendered
13      }
14    }
```

El archivo `~/terraform/files/debian.tpl` debe contener la información de *cloud init*. Si el archivo o la carpeta que lo contiene no existen, se pueden ejecutar los siguientes comandos:

```
1    mkdir ~/terraform/files/ && cd ~/terraform/files/
2    touch debian.tpl
```

Luego, en el archivo `debian.tpl` creado se agrega la información de *cloud init*. Con esto, Terraform instancia la VM considerando esa receta de creación. El resto del archivo `instance.tf` contiene la información necesaria para activar el paquete de exportación de métricas hacia Prometheus, crear las configuraciones de red necesarias y habilitar puertos en el grupo de seguridad. Con el archivo creado, en la carpeta `~/terraform/openstack` se ejecutan los siguientes comandos:

```
1    terraform init
2    terraform apply
```

Una vez finalice, en pantalla debería aparecer el mensaje `Apply complete Resources: 9 added, 0 changed, 0 destroyed` indicando que la VM fue instanciada correctamente. Para hacer una doble validación, se accede a la interfaz gráfica de OpenStack y se valida que efectivamente la instancia fue creada, como se muestra en la Figura **2-5**.

Como último paso, la instancia puede ser accedida vía SSH usando la IP flotante asignada por OpenStack (en la Figura **2-5** se observa que es `10.20.20.167`) y la contraseña configurada en el archivo *cloud init* (en este caso `debian`). Es importante mencionar que al tratarse de una VM posee un OS completo, por lo cual, cualquier comando Linux/Debian puede ejecutarse. A manera de prueba, se valida que el paquete de exportación de métricas a Prometheus está activo mediante el comando `ps aux | grep -i prometh`, ejecutado dentro de la VM.

E. Anexo: Pseudocódigo para la implementación del algoritmo

El pseudocódigo planteado en este anexo, implementa el algoritmo cuyo diagrama de flujo es propuesto en la Figura **4-2**. Antes de empezar el recorrido paso a paso por el pseudocódigo, es importante considerar que, basándose en la Tabla **4-1**, `x[t]` representa la medición general de alguna variable en el instante de tiempo actual. Por otro lado `pd_x` representa la predicción de esa variables. Dado que el algoritmo prioriza el autoescalamiento proactivo sobre el reactivo, se consideran las posibles combinaciones entre valor actual y el valor predicho sobre los umbrales definidos.

El algoritmo inicia con una verificación de la existencia de datos para evaluar el estado actual del sistema. Si no hay datos, se opta por no seguir el procedimiento y reintentar hasta que existan. En caso de que existan datos, el siguiente paso es consultar cuál es el estado actual para evaluar si debe tomarse una decisión de autoescalamiento. Por consiguiente, se consulta si alguna de las variables ha superado el umbral superior, dado que infiere una posible saturación o congestión del sistema. El procedimiento, en caso de que se supere el umbral inferior, es idéntico. Si alguna de las variables supera el umbral, se evalúa cuál es el valor de la predicción en ese instante; si la predicción asegura que en el siguiente instante también se viola el umbral, se toma la decisión de autoescalamiento asumiendo que el sistema se saturará o hará uso ineficiente de recursos, según sea el caso. Si la predicción no asegura la violación del umbral, entonces se opta por esperar el siguiente instante de tiempo y evaluar nuevamente el estado. Si en este nuevo instante de tiempo alguna de las variables viola el umbral, se toma la decisión de autoescalar y se asume que la predicción anterior fue errónea. En caso contrario, se considera un nuevo valor de predicción; si esta nueva predicción no asegura violación del umbral, entonces el proceso vuelve a la primera tarea de verificación de los datos porque se asume que el sistema no se saturará. En caso de que la nueva predicción asegure la violación del umbral, entonces se espera al siguiente instante de tiempo para evaluar el estado del sistema. Si, en este nuevo instante, hay violación de umbral, entonces se toma la decisión de autoescalamiento, confirmando la predicción. En caso contrario, el proceso vuelve a la primera tarea dado que el sistema no se saturó. Es importante mencionar que después de cualquier decisión de autoescalamiento, el procedimiento vuelve a la primera tarea de verificación de datos.

Por otra parte, si durante la evaluación del estado del sistema no se supera alguno de los umbrales establecidos, se asume que el sistema mantiene un comportamiento normal o dentro de los límites definidos. Aun con este comportamiento, se consideran el valor predicho para alguna de las variables. Si este valor infiere una posible saturación en el siguiente instante de tiempo, se retoma la evaluación del estado del sistema. En caso contrario, se mantiene el proceso de monitoreo de la disponibilidad de los datos. A continuación se presenta el pseudocódigo que implementa lo descrito.

```
inicio;
t = 0;
n_instances = 1;
variables = obtener(datos);
while(true) {
    if(variables != NULL) {
        print "Hay datos disponibles.";
        while(variables != NULL) {
            if(x[t] > threshold_x_max) {
                if(pd_x > threshold_x_max) {
                    funcion(autoescalar);
                    print "El sistema se saturara.";
                    n_instances = n_instances + 1;
                }
                else {
                t = t + 1;
                if(x[t] > threshold_x_max) {
                    funcion(autoescalar);
                    print "Prediccion erronea. El sistema se saturara.";
                    n_instances = n_instances + 1;
                }
                else {
                    if(pd_x > threshold_x_max) {
                        t = t + 1;
                        if(x[t] > threshold_x_max) {
                            funcion(autoescalar);
                            print "Prediccion confirmada. El sistema se saturara
    .";
                            n_instances = n_instances + 1;
                        }
                        else {
                            t = t + 1;
                            print "El sistema no se saturo.";
                        }
```

```
34                  }
35                  else {
36                      t = t + 1;
37                      print "El sistema no se saturara.";
38                  }
39              }
40           }
41          }
42          else {
43          if(x[t] < threshold_x_min) {
44               if(pd_x < threshold_x_min) {
45                   if(n_instances == 1) {
46                       print "No se pueden eliminar instancias. Se sugiere
    reducir los recursos de la instancia.";
47                   }
48                   else {
49                       funcion(autoescalar);
50                       print "El sistema se sobredimensionara.";
51                       n_instances = n_instances - 1;
52                   }
53               }
54               else {
55                   t = t + 1;
56                   if(x[t] < threshold_x_min) {
57                       if(n_instances == 1) {
58                           print "No se pueden eliminar instancias. Se sugiere
    reducir los recursos de la instancia.";
59                       }
60                       else {
61                           funcion(autoescalar);
62                           print "El sistema se sobredimensionara.";
63                           n_instances = n_instances - 1;
64                       }
65                   }
66                   else {
67                       if(pd_x < threshold_x_min) {
68                           t = t + 1;
69                           if(x[t] < threshold_x_min) {
70                               if(n_instances == 1) {
71                                   print "No se pueden eliminar instancias. Se
    sugiere reducir los recursos de la instancia.";
72                               }
```

```
                                else {
                                    funcion(autoescalar);
                                    print "El sistema se sobredimensionara.";
                                    n_instances = n_instances - 1;
                                }
                            }
                            else {
                                t = t + 1;
                                print "El sistema no se sobredimensiono.";
                            }
                        }
                        else {
                            t = t + 1;
                            print "El sistema no se sobredimensionara.";
                        }
                    }
                }
            }
            else {
                if(pd_x > threshold_x_max || pd_x < threshold_x_min) {
                    t = t + 1;
                    print "El sistema puede superar alguno de los umbrales.";
                }
                else {
                    t = t + 1;
                    print "Mantiene el comportamiento normal.";
                }
             }
        }
    }
    }
    else {
        print "Reintentando toma de datos.";
        t = t + 1;
    }
}
```

Por último, existe una condición de disponibilidad que impide desescalar instancias cuando solo existe una en el sistema. La razón es que, claramente, si se desescala la única instancia que brinda el servicio se genera una condición de indisponibilidad. En lugar de desescalar la instancia, se recomienda una reducción en los recursos asignados (escalamiento vertical) sin ejecutar el cambio.

Bibliografía

[1] P. P. Marino, M. Garrich, and F. J. M. Muro, "The role of open-source network optimization software in the SDN/NFV World," in *Optics InfoBase Conference Papers*, vol. Part F84-O, (Washington, D.C.), p. Th1D.1, OSA, mar 2018.

[2] T. Shuminoski and T. Janevski, "5G mobile terminals with advanced QoS-based user-centric aggregation (AQUA) for heterogeneous wireless and mobile networks," *Wireless Networks*, vol. 22, pp. 1553–1570, jul 2016.

[3] Y. Wang, P. Li, L. Jiao, Z. Su, N. Cheng, X. S. Shen, and P. Zhang, "A Data-Driven Architecture for Personalized QoE Management in 5G Wireless Networks," *IEEE Wireless Communications*, vol. 24, pp. 102–110, feb 2017.

[4] Ericcson, "Ericsson Mobility Report," *Ericsson*, no. June, p. 36, 2020.

[5] M. Garrich, F. J. Moreno-Muro, M. V. Bueno Delgado, and P. P. Mariño, "Open-Source Network Optimization Software in the Open SDN/NFV Transport Ecosystem," *Journal of Lightwave Technology*, vol. 37, pp. 75–88, jan 2019.

[6] S. Akhshabi, L. Anantakrishnan, A. C. Begen, and C. Dovrolis, "What happens when HTTP adaptive streaming players compete for bandwidth?," in *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video - NOSSDAV '12*, (New York, New York, USA), p. 9, ACM Press, 2013.

[7] P. Tantisarkhornkhet and W. Werapun, "QLB: QoS routing algorithm for Software-Defined Networking," in *2016 International Symposium on Intelligent Signal Processing and Communication Systems, ISPACS 2016*, pp. 1–6, IEEE, oct 2017.

[8] T. F. Yu, K. Wang, and Y. H. Hsu, "Adaptive routing for video streaming with QoS support over SDN networks," in *International Conference on Information Networking*, vol. 2015-Janua, pp. 318–323, IEEE, jan 2015.

[9] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks," in *2012 Conference Handbook - Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, APSIPA ASC 2012*, (Hollywood, CA, USA), pp. 1–8, Asia-Pacific Signal and Information Processing

Association, 2012 Annual Summit and Conference International Organizing Committee, 2012.

[10] C. Hu, Q. Wang, and X. Dai, "SDN over IP: Enabling Internet to Provide Better QoS Guarantee," in *Proceedings - 2015 9th International Conference on Frontier of Computer Science and Technology, FCST 2015*, pp. 46–51, IEEE, aug 2015.

[11] V. G. Vassilakis, I. D. Moscholios, and M. D. Logothetis, "Quality of service differentiation in heterogeneous CDMA networks: a mathematical modelling approach," *Wireless Networks*, vol. 24, pp. 1279–1295, may 2018.

[12] R. Alvizu, G. Maier, S. Troia, V. M. Nguyen, and A. Pattavina, "SDN-based network orchestration for new dynamic Enterprise Networking services," in *International Conference on Transparent Optical Networks*, pp. 1–4, IEEE, jul 2017.

[13] S. Jain, M. Khandelwal, A. Katkar, and J. Nygate, "Applying big data technologies to manage QoS in an SDN," in *2016 12th International Conference on Network and Service Management, CNSM 2016 and Workshops, 3rd International Workshop on Management of SDN and NFV, ManSDN/NFV 2016, and International Workshop on Green ICT and Smart Networking, GISN 2016*, pp. 302–306, IEEE, oct 2017.

[14] M. Shamseddine, I. Elhajj, A. Chehab, and A. Kayssi, "A virtual QoS-Adaptive network connectivity service: An SDN approach," in *2016 IEEE International Multidisciplinary Conference on Engineering Technology, IMCET 2016*, pp. 92–96, IEEE, nov 2016.

[15] J. W. Kleinrouweler, S. Cabrero, and P. Cesar, "Delivering stable high-quality video," in *Proceedings of the 7th International Conference on Multimedia Systems - MMSys '16*, (New York, New York, USA), pp. 1–10, ACM Press, 2016.

[16] A. A. Barakabitze, L. Sun, I. H. Mkwawa, and E. Ifeachor, "A Novel QoE-Centric SDN-Based Multipath Routing Approach for Multimedia Services over 5G Networks," in *IEEE International Conference on Communications*, vol. 2018-May, pp. 1–7, IEEE, may 2018.

[17] H. Sinha, G. Raj, and T. Choudhury, "Computing an adaptive mVoIP Services through SDN networks," in *Proceedings of the 5th International Conference on System Modeling and Advancement in Research Trends, SMART 2016*, pp. 170–174, IEEE, 2017.

[18] K. T. Bagci, K. E. Sahin, and A. M. Tekalp, "Queue-allocation optimization for adaptive video streaming over software defined networks with multiple service-levels," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 1519–1523, IEEE, sep 2016.

[19] Z. Xu, W. Liang, A. Galis, and Y. Ma, "Throughput maximization and resource optimization in NFV-enabled networks," in *IEEE International Conference on Communications*, pp. 1–7, IEEE, may 2017.

[20] 3GPP, "Telecommunication management; Study on the Self-Organizing Networks (SON) for 5G networks," tech. rep., 3GPP, 2019.

[21] Z. Zhou, T. Zhang, and A. Kwatra, "NFV Closed-loop Automation Experiments using Deep Reinforcement Learning," in *INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2019*, pp. 696–701, Institute of Electrical and Electronics Engineers Inc., apr 2019.

[22] A. India, M. Sandilya, and N. Pd, "Zero Touch SDN NFV," tech. rep., 2018.

[23] I. G. Ben Yahia, J. Bendriss, A. Samba, and P. Dooze, "CogNitive 5G networks: Comprehensive operator use cases with machine learning for management operations," in *Proceedings of the 2017 20th Conference on Innovations in Clouds, Internet and Networks, ICIN 2017*, pp. 252–259, IEEE, mar 2017.

[24] A. Ben Letaifa, "Adaptive QoE monitoring architecture in SDN networks: Video streaming services case," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 1383–1388, IEEE, jun 2017.

[25] P. Torres, P. Marques, H. Marques, R. Dionisio, T. Alves, L. Pereira, and J. Ribeiro, "Data analytics for forecasting cell congestion on LTE networks," in *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pp. 1–6, IEEE, jun 2017.

[26] P. Torres, H. Marques, P. Marques, and J. Rodriguez, "Using Deep Neural Networks for Forecasting Cell Congestion on LTE Networks: A Simple Approach," pp. 276–286, Springer, Cham, 2018.

[27] A. Samba, Y. Busnel, A. Blanc, P. Dooze, and G. Simon, "Instantaneous throughput prediction in cellular networks: Which information is needed?," in *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, pp. 624–627, IEEE, may 2017.

[28] N. Bui, F. Michelinakis, and J. Widmer, "A model for throughput prediction for mobile users," in *20th European Wireless Conference, EW 2014*, (Barcelona), pp. 1–6, VDe, 2014.

[29] B. Li, W. Lu, S. Liu, and Z. Zhu, "Deep-learning-assisted network orchestration for on-demand and cost-effective VNF service chaining in inter-DC elastic optical networks," *Journal of Optical Communications and Networking*, vol. 10, pp. D29–D41, oct 2018.

[30] A. Nadjaran Toosi and R. Buyya, "Acinonyx: Dynamic Flow Scheduling for Virtual Machine Migration in SDN-Enabled Clouds," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, pp. 886–894, IEEE, dec 2018.

[31] Google, "Ajuste de escala automático de grupos de instancias." https://cloud.google.com/compute/docs/autoscaler/, 2021.

[32] OpenStack Project, "Open Source Cloud Computing Systems." https://www.openstack.org/, 2011.

[33] H. Niu, C. Li, A. Papathanassiou, and G. Wu, "RAN architecture options and performance for 5G network evolution," in *2014 IEEE Wireless Communications and Networking Conference Workshops, WCNCW 2014*, pp. 294–298, Institute of Electrical and Electronics Engineers Inc., oct 2014.

[34] F. Marzouk, J. P. Barraca, and A. Radwan, "On Energy Efficient Resource Allocation in Shared RANs: Survey and Qualitative Analysis," *IEEE Communications Surveys and Tutorials*, vol. 22, pp. 1515–1538, jul 2020.

[35] M. J. Scheepers, "Virtualization and Containerization of Application Infrastructure : A Comparison," 2014.

[36] E. G. Radhika, G. S. Sadasivam, and J. F. Naomi, "An efficient predictive technique to autoscale the resources for web applications in private cloud," in *Proceedings of the 4th IEEE International Conference on Advances in Electrical and Electronics, Information, Communication and Bio-Informatics, AEEICB 2018*, Institute of Electrical and Electronics Engineers Inc., oct 2018.

[37] Prometheus, "Prometheus - Monitoring system & time series database." https://prometheus.io/, 2017.

[38] Grafana, "Grafana: The open observability plataform." https://grafana.com/, 2020.

[39] HashiCorp, "Introduction - Terraform by HashiCorp." https://www.terraform.io/intro/index.html#execution-plans, 2020.

[40] A. Levin, D. Lorenz, G. Merlino, A. Panarello, A. Puliafito, and G. Tricomi, "Hierarchical load balancing as a service for federated cloud networks," *Computer Communications*, vol. 129, pp. 125–137, sep 2018.

[41] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Providing Performance Guarantees for Cloud-Deployed Applications," *IEEE Transactions on Cloud Computing*, vol. 8, pp. 269–281, jan 2020.

[42] V. Simic, B. Stojanovic, and M. Ivanovic, "Optimizing the performance of optimization in the cloud environment–An intelligent auto-scaling approach," *Future Generation Computer Systems*, vol. 101, pp. 909–920, dec 2019.

[43] L. Phan and K. Liu, "OpenStack Network Acceleration Scheme for Datacenter Intelligent Applications," in *IEEE International Conference on Cloud Computing, CLOUD*, vol. 2018-July, pp. 962–965, IEEE Computer Society, sep 2018.

[44] B. Sniezynski, P. Nawrocki, M. Wilk, M. Jarzab, and K. Zielinski, "VM Reservation Plan Adaptation Using Machine Learning in Cloud Computing," *Journal of Grid Computing*, vol. 17, pp. 797–812, dec 2019.

[45] L. Gavrilovska, V. Rakovic, and D. Denkovski, "Aspects of Resource Scaling in 5G-MEC: Technologies and Opportunities," in *2018 IEEE Globecom Workshops, GC Wkshps 2018 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., feb 2019.

[46] J. Zhang, F. Ren, and C. Lin, "Survey on transport control in data center networks," *IEEE Network*, vol. 27, no. 4, pp. 22–26, 2013.

[47] W. Hajji, T. A. Genez, F. P. Tso, L. Cui, and I. Phillips, "Dynamic Network Function Chain Composition for Mitigating Network Latency," in *Proceedings - IEEE Symposium on Computers and Communications*, vol. 2018-June, pp. 316–321, Institute of Electrical and Electronics Engineers Inc., nov 2018.

[48] S. Hykes, "Empowering App Development for Developers — Docker." https://www.docker.com/, 2013.

[49] Kubernetes, "Kubernetes Documentation - Kubernetes." https://kubernetes.io/docs/home/, 2019.

[50] S. Kho Lin, U. Altaf, G. Jayaputera, J. Li, D. Marques, D. Meggyesy, S. Sarwar, S. Sharma, W. Voorsluys, R. Sinnott, A. Novak, V. Nguyen, and K. Pash, "Auto-Scaling a Defence Application across the Cloud Using Docker and Kubernetes," in *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018*, pp. 327–334, Institute of Electrical and Electronics Engineers Inc., jan 2019.

[51] B. Zurkowski and K. Zielinski, "Towards Self-Organizing Cloud Polyglot Database Systems," in *International Conference on Self-Adaptive and Self-Organizing Systems, SASO*, vol. 2019-June, pp. 82–87, IEEE Computer Society, jun 2019.

[52] C. Organisations and P. Date, "Network Functions Virtualisation (NFV)," no. 1, pp. 1–20, 2015.

[53] W. Nakimuli, J. Garcia-Reinoso, B. Nogales, I. Vidal, D. Gomes, and D. Lopez, "Reducing Service Creation Time Leveraging on Network Function Virtualization," *IEEE Access*, vol. 8, pp. 155679–155696, 2020.

[54] Canonical, "MicroStack - OpenStack in a snap." https://microstack.run/, 2020.

[55] Gnocchi Project, "Gnocchi – Metric as a Service." https://gnocchi.xyz/stable_4.2/index.html.

[56] T. T. Hoang, M. T. Tao, and P. H. Au, "Research and implementation of monitoring systems Prometheus and Grafana," 2020.

[57] Grafana Labs, "OpenStack Dashboard dashboard for Grafana." https://grafana.com/grafana/dashboards/9701, 2018.

[58] Open Source MANO, "White papers, Scope, Functionality, Operation and Integration Guidelines," no. 1, pp. 1–44, 2019.

[59] The Linux Foundation, "ONAP." https://www.onap.org/, 2021.

[60] M. Gilbert, *Artificial Intelligence for Autonomous Networks*. 2018.

[61] L. Chen, D. Yang, D. Zhang, C. Wang, J. Li, and T. M. T. Nguyen, "Deep mobile traffic forecast and complementary base station clustering for C-RAN optimization," *Journal of Network and Computer Applications*, vol. 121, pp. 59–69, 2018.

[62] L. Jorguseski, A. Pais, F. Gunnarsson, A. Centonza, and C. Willcock, "Self-organizing networks in 3GPP: Standardization and future trends," *IEEE Communications Magazine*, vol. 52, pp. 28–34, dec 2014.

[63] S. Ahmad and A. H. Mir, "Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers," *Journal of Network and Systems Management*, vol. 29, pp. 1–59, jan 2021.

[64] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *Neural Computation*, vol. 28, pp. 2474–2504, nov 2016.

[65] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, nov 2017.

[66] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth Estimation: Metrics, Measurement Techniques, and Tools," vol. 17, pp. 27–35, nov 2003.

[67] S. S. Chaudhari and R. C. Biradar, "Survey of Bandwidth Estimation Techniques in Communication Networks," vol. 83, pp. 1425–1476, jul 2015.

[68] J. Xu, L. Tang, Q. Chen, and L. Yi, "Study on based reinforcement Q-Learning for mobile load balancing techniques in LTE-A HetNets," in *Proceedings - 17th IEEE International Conference on Computational Science and Engineering, CSE 2014, Jointly with 13th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2014, 13th International Symposium on Pervasive Systems,*, pp. 1766–1771, 2015.

[69] S. S. Mwanje and A. Mitschele-Thiel, "A Q-learning strategy for LTE mobility Load Balancing," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, pp. 2154–2158, 2013.

[70] J. J. Montano Moreno, A. Palmer Pol, and P. Munoz Gracia, "Artificial neural networks applied to forecasting time series.," *Psicothema*, vol. 23, no. 2, pp. 322–9, 2011.

[71] H. D. Trinh, L. Giupponi, and P. Dini, "Mobile Traffic Prediction from Raw Data Using LSTM Networks," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, vol. 2018-Septe, pp. 1827–1832, Institute of Electrical and Electronics Engineers Inc., dec 2018.

[72] S. Cao and W. Liu, "LSTM Network Based Traffic Flow Prediction for Cellular Networks," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, pp. 643–653, Springer, 2019.

[73] M. Algorri Álvarez, "Caracterización de tecnologías de procesamiento de datos en streaming sobre una arquitectura orientada al dato," Master's thesis, 10 2018.

[74] R. Kempter, W. Gerstner, and J. L. van Hemmen, "Hebbian learning and spiking neurons," *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, vol. 59, pp. 4498–4514, apr 1999.

[75] Y. Cui, C. Surpur, S. Ahmad, and J. Hawkins, "A comparative study of HTM and other neural network models for online sequence learning with streaming data," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2016-Octob, pp. 1530–1538, Institute of Electrical and Electronics Engineers Inc., oct 2016.

[76] J. Struye and S. Latré, "Hierarchical temporal memory and recurrent neural networks for time series prediction: An empirical validation and reduction to multilayer perceptrons," *Neurocomputing*, vol. 396, pp. 291–301, jul 2020.

[77] E. Nugamanov and A. I. Panov, "Hierarchical Temporal Memory with Reinforcement Learning," in *Procedia Computer Science*, vol. 169, pp. 123–131, Elsevier B.V., jan 2020.

[78] M. Mdini, *Anomaly detection and root cause diagnosis in cellular networks*. PhD thesis, 2019.

[79] C. Wang, Z. Zhao, L. Gong, L. Zhu, Z. Liu, and X. Cheng, "A Distributed Anomaly Detection System for In-Vehicle Network Using HTM," *IEEE Access*, vol. 6, pp. 9091–9098, jan 2018.

[80] A. Barua, D. Muthirayan, P. P. Khargonekar, and M. A. Al Faruque, "Hierarchical Temporal Memory Based Machine Learning for Real-Time, Unsupervised Anomaly Detection in Smart Grid: WiP Abstract," in *Proceedings - 2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems, ICCPS 2020*, pp. 188–189, Institute of Electrical and Electronics Engineers Inc., apr 2020.

[81] K. Zhang, F. Zhao, S. Luo, Y. Xin, H. Zhu, and Y. Chen, "Online intrusion scenario discovery and prediction based on hierarchical temporal memory (HTM)," *Applied Sciences (Switzerland)*, vol. 10, p. 2596, apr 2020.

[82] Numenta, "HIERARCHICAL TEMPORAL MEMORY. HTM Cortical Learning Algorithms." http://www.numenta.com/faq.html#cla_paper, 2011.

[83] Numenta, "HTM School." https://numenta.org/htm-school/, 2020.

[84] S. Purdy, "Encoding Data for HTM Systems," feb 2016.

[85] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin, *Biological and Machine Intelligence (BAMI)*. 2016.

[86] G. K. Karagiannidis and A. S. Lioumpas, "An improved approximation for the Gaussian Q-function," *IEEE Communications Letters*, vol. 11, pp. 644–646, aug 2007.

[87] Numenta, "Numenta, Where Neuroscience Meets Machine Intelligence." https://numenta.com/, 2020.

[88] "htm-community/htm.core: Actively developed Hierarchical Temporal Memory (HTM) community fork (continuation) of NuPIC. Implementation for C++ and Python." https://github.com/htm-community/htm.core.

[89] D. F. Rueda, D. Vergara, and D. Reniz, "Big Data Streaming Analytics for QoE Monitoring in Mobile Networks: A Practical Approach," in *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, pp. 1992–1997, Institute of Electrical and Electronics Engineers Inc., jan 2019.

[90] N. Salhab, S. E. Falou, R. Rahim, S. E. E. Ayoubi, and R. Langar, "Optimization of the implementation of network slicing in 5G RAN," in *2018 IEEE Middle East and North Africa Communications Conference, MENACOMM 2018*, pp. 1–6, Institute of Electrical and Electronics Engineers Inc., jun 2018.

[91] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," *IEEE Communications Magazine*, vol. 55, pp. 80–87, may 2017.

[92] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Network function virtualization in 5G," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 84–91, 2016.

[93] D. Lee, J. H. Yoo, and J. W. K. Hong, "Deep Q-networks based auto-scaling for service function chaining," in *16th International Conference on Network and Service Management, CNSM 2020, 2nd International Workshop on Analytics for Service and Application Management, AnServApp 2020 and 1st International Workshop on the Future Evolution of Internet Protocols, IPFutu*, 2020.

[94] R. Ranjan, B. Benatallah, S. Dustdar, and M. P. Papazoglou, "Cloud Resource Orchestration Programming: Overview, Issues, and Directions," vol. 19, pp. 46–56, sep 2015.

[95] S. Becker, G. Brataas, and S. Lehrig, *Engineering Scalable, Elastic, and Cost-Efficient Cloud Computing Applications*. Springer International Publishing, 2017.

[96] Open Source MANO, "OSM Autoscaling." https://osm.etsi.org/wikipub/index.php/, 2019.

[97] T. Choi, T. Kim, W. Tavernier, A. Korvala, and J. Pajunpää, "Agile Management and Interoperability Testing of SDN/NFV-Enriched 5G Core Networks:," *ETRI Journal*, vol. 40, pp. 72–88, feb 2018.

[98] Y. Ren, T. Phung-Duc, J. C. Chen, and Z. W. Yu, "Dynamic auto scaling algorithm (DASA) for 5G mobile networks," in *2016 IEEE Global Communications Conference, GLOBECOM 2016 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., 2016.

[99] Y. Ren, T. Phung-Due, Y. K. Liu, J. C. Chen, and Y. H. Lin, "ASA: Adaptive VNF Scaling Algorithm for 5G Mobile Networks," in *Proceedings of the 2018 IEEE 7th International Conference on Cloud Networking, CloudNet 2018*, Institute of Electrical and Electronics Engineers Inc., nov 2018.

[100] P. C. Amogh, G. Veeramachaneni, A. K. Rangisetti, B. R. Tamma, and F. A. Antony, "A cloud native solution for dynamic auto scaling of MME in LTE," in *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, vol. 2017-Octob, pp. 1–7, Institute of Electrical and Electronics Engineers Inc., feb 2018.

[101] Y. T. Lee, H. L. Chao, and J. W. Tang, "Scalable and elastic cloud data center for self-organizing dense small cell networks," in *17th Asia-Pacific Network Operations and Management Symposium: Managing a Very Connected World, APNOMS 2015*, pp. 420–423, Institute of Electrical and Electronics Engineers Inc., sep 2015.

[102] S. Khairi, B. Raouyane, and M. Bellafkih, "Novel QoE monitoring and management architecture with eTOM for SDN-based 5G networks: SLA verification scenario," *Cluster Computing*, vol. 23, pp. 1–12, feb 2020.

[103] I. Afolabi, J. Prados-Garzon, M. Bagaa, T. Taleb, and P. Ameigeiras, "Dynamic resource provisioning of a scalable E2E network slicing orchestration system," *IEEE Transactions on Mobile Computing*, vol. 19, pp. 2594–2608, nov 2020.

[104] M. M. Rahman, C. Despins, and S. Affes, "Design Optimization of Wireless Access Virtualization Based on Cost & QoS Trade-Off Utility Maximization," *IEEE Transactions on Wireless Communications*, vol. 15, pp. 6146–6162, sep 2016.

[105] L. Tang, X. He, X. Yang, Y. Wei, X. Wang, and Q. Chen, "ARMA-Prediction-Based Online Adaptive Dynamic Resource Allocation in Wireless Virtualized Network," *IEEE Access*, vol. 7, pp. 130438–130450, 2019.

[106] S. Rizou, P. Athanasoulis, P. Andriani, F. Iadanza, G. Carrozzo, D. Breitgand, A. Weit, D. Griffin, D. Jimenez, U. Acar, and O. P. Gordo, "A Service Platform Architecture Enabling Programmable Edge-To-Cloud Virtualization for the 5G Media Industry," in *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB*, vol. 2018-June, IEEE Computer Society, aug 2018.

[107] "Introducing AWS Lambda." https://aws.amazon.com/es/about-aws/whats-new/2014/11/13/introducing-aws-lambda/, 2014.

[108] "5G-PPP." https://5g-ppp.eu/, 2021.

[109] A. H. Ghorab, A. Kusedghi, M. A. Nourian, and A. Akbari, "Joint VNF Load Balancing and Service Auto-Scaling in NFV with Multimedia Case Study," in *2020 25th International Computer Conference, Computer Society of Iran, CSICC 2020*, Institute of Electrical and Electronics Engineers Inc., jan 2020.

[110] A. Kusedghi, A. Ghorab, and A. Akbari, "XeniumNFV: A unified, dynamic, distributed and event-driven SDN/NFV testbed," in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, vol. 2018-Decem, pp. 320–326, IEEE Computer Society, dec 2018.

[111] F. B. Anacona and K. T. Tobar, *Scalability Analysis of LTE-EPC in an NFV Environment*. PhD thesis, 2018.

[112] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," *Journal of Grid Computing*, vol. 12, pp. 559–592, nov 2014.

[113] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling," in *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2017*, pp. 64–73, Institute of Electrical and Electronics Engineers Inc., jul 2017.

[114] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, and S. Sioutas, "Dependable horizontal scaling based on probabilistic model checking," in *Proceedings - 2015 IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015*, pp. 31–40, Institute of Electrical and Electronics Engineers Inc., jul 2015.

[115] A. A. Neghabi, N. J. Navimipour, M. Hosseinzadeh, and A. Rezaee, "Load Balancing Mechanisms in the Software Defined Networks: A Systematic and Comprehensive Review of the Literature," vol. 6, pp. 14159–14178, 2018.

[116] S. Dutta, T. Taleb, and A. Ksentini, "QoE-aware elasticity support in cloud-native 5G systems," in *2016 IEEE International Conference on Communications, ICC 2016*, Institute of Electrical and Electronics Engineers Inc., jul 2016.

[117] Apache Software Foundation, "The Apache HTTP Server Project." https://httpd.apache.org/, 2021.

[118] Apache Software Foundation, "ab [Apache HTTP server benchmarking tool]." https://httpd.apache.org/docs/2.4/programs/ab.html, 2014.

[119] GitHub, "caprivm/thesis_msc Wiki." https://github.com/caprivm/thesis_msc/wiki, 2021.

[120] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.